ISSN 0265-2919

80p 8

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

APPLICATION

RAISING THE STANDARD Japanese and American manufacturers have united to produce MSX, a common standard — 141

HARDWARE

VIDEO STAR We examine the Spectravideo, a computer that incorporates many of the MSX standard features — 149

SOFTWARE

COMPANY REPORTS We continue our detailed look at three business packages — 152

COMPUTER SCIENCE

CHANGING LINES We design a simple decoder circuit from first principles — 146

PROGRAMMING PROJECTS

A DIFFERENT ANGLE The first part of a series explaining mathematics in BASIC — 154

JARGON

FROM BANDWIDTH TO BASE A weekly glossary of computing terms — 148

MACHINE CODE

STARTING ORDERS We look at the instructions needed to assemble a program — 156

PROFILE

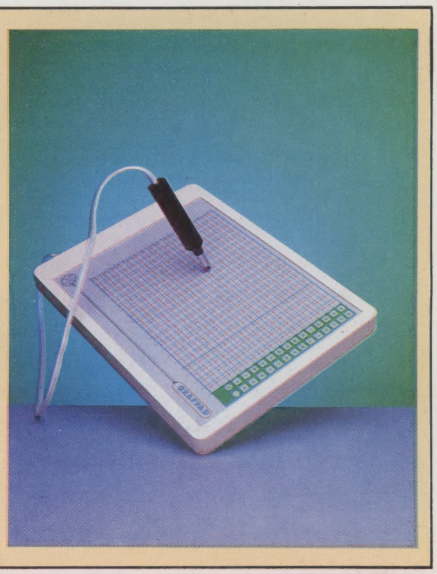
LEADING EDGE SORD is one of the smaller Japanese companies to break into the market — 159

WORKSHOP

SWITCH BOARD We construct a simple switching circuit using AND and OR gates — 144

Next Week

- **The Grafpad is an exciting low cost digitiser for BBC, Spectrum and Commodore 64 users. We try out the system – from simple doodles to computer aided design.**
- **We look at the evolution and history of microprocessors.**
- **The Workshop series moves on to more ambitious projects as we build a half adder logic circuit from simple components.**



COMING VERY SOON

FREE

MACHINE CODE MONITOR AND ASSEMBLER PROGRAM ON CASSETTE

Written specially for Home Computer Advanced Course readers

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

Editor Max Phillips; **Art Director** David Whelan; **Production Editor** Catherine Cardwell; **Staff Writer** Brian Morris; **Picture Editor** Claudia Zeff; **Designers** Hazel Bennington, Julian Dorr; **Sub Editor** Robert Pickering; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Lisa Kelly, Steven Colwill, Gareth Jefferson, Geoff Bains, Tony Harrington, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

RAISING THE STANDARD

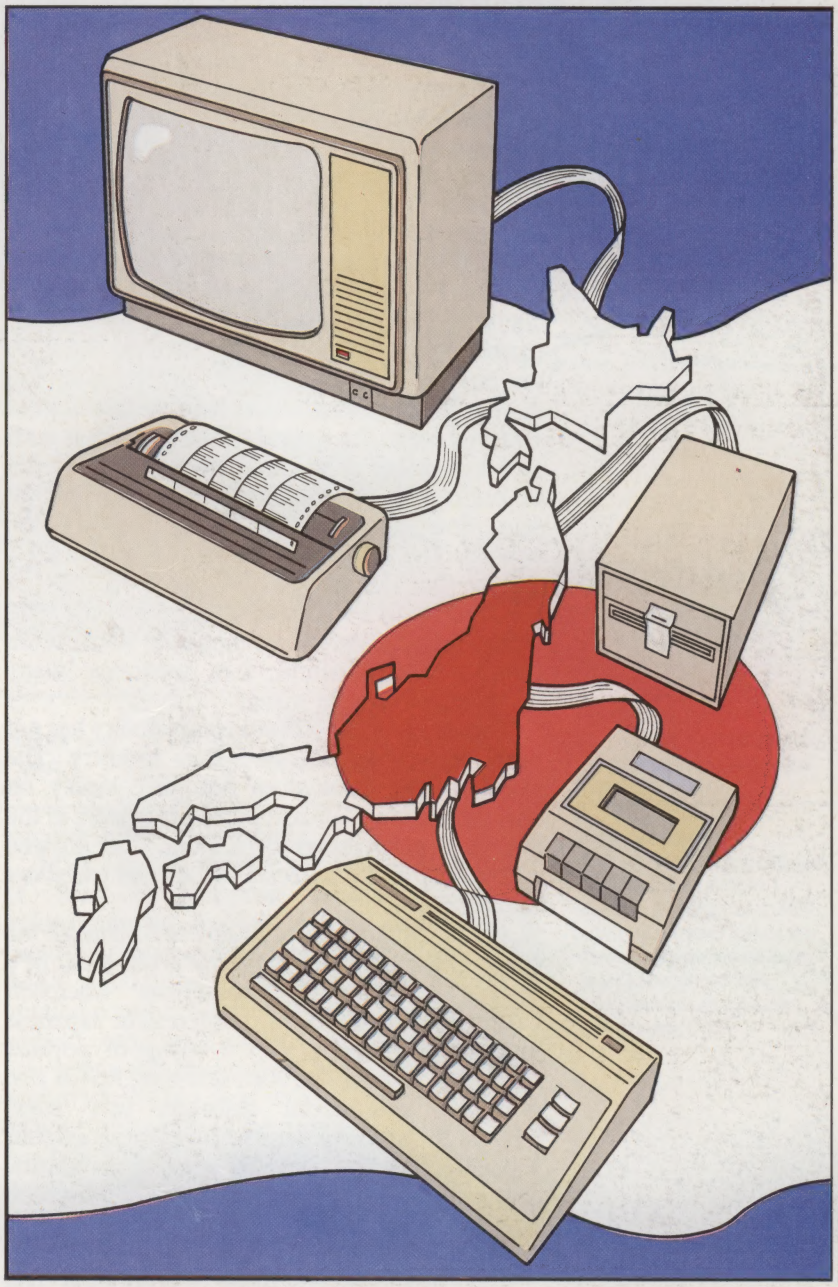
One of the problems facing anyone writing software for a home computer is that of compatibility. A program written for the Spectrum will not run on any other home computer. The MSX standard is an attempt by a group of manufacturers to provide a general compatibility of hardware and software between machines.

To understand why different home computers are so idiosyncratic in design, we need to be aware of how the microcomputer industry developed. The first microprocessors to make a big impact on the home computer market were Intel's 8080 and, arguably, Motorola's 6800. The instruction sets of these processors were fixed from the beginning. The compatibility problem arose from the processors' versatility. For example, sending a byte of data to an output port involved exactly the same instruction, whatever the computer, provided the same processor was used. But the output could have any one of hundreds or thousands of addresses. The early home micros were made by numerous garage and back yard manufacturers, so naturally they had no agreement between them on where the input or output ports should be located in the memory map. Manufacturer A might choose the parallel printer output port to be located at address 255, while manufacturer B might choose address 254.

This situation was bad enough, but with the advent of screen graphics controlled by special CRT controller chips, and sound effects controlled by dedicated sound chips, the situation became even more chaotic. Any program taking advantage of the special capabilities of any one computer could be guaranteed not to run on another computer without considerable rewriting.

Had there been one company in a powerful enough position to enforce a standard from the beginning, the situation might have been entirely different. But it wasn't like that. In the early days of home computers, there were numerous small manufacturers, each with a different house style and standard. Not only were the machines physically and electronically different from each other, even the programming languages supplied with them were incompatible with other machines. From the beginning, BASIC never enjoyed a standard. It was not, in the 1970s at least, taken very seriously by the professional computing community, who saw it as being strictly a beginner's language.

During the late 1970s and early 1980s, microcomputers developed at a tremendous pace.



TONY DUNCAN-SMITH

Road To Success
The MSX standard is Japan's route to the world's computer markets. If it is successful, Japan could dominate the computer market in the same way it has succeeded in the hi-fi and camera business

Pioneering designs such as the Apple started to incorporate refinements like sophisticated graphics and colour. To make these innovations usable, the manufacturers had to develop their own dialects of BASIC and so versions of the language proliferated.

The price of this proliferation is not so much a greater choice for the consumer, as frustration for owners, manufacturers and software writers alike. A proud SORD owner might be desperate to get his hands on Ju-Ju The Unobtainable or whatever the latest game craze is. If the program was written

for the Spectrum, however, he might have to wait quite a while before a version becomes available for his machine.

If you're a computer manufacturer with a new product to launch, you know that there will be almost no software for your product until a substantial user base has been established. With little or no software to go with the computer, you will be well aware that the sales appeal of your machine will be strictly limited and this could risk all your investment in developing the product.

If you earn your living by writing commercial software, your sales will be limited to (at best) the number of people who own the model of computer the program has been written for. Suppose you have created an adventure game called The Dungeons of Rathbone, replete with a Faceless Presence, a sadistic Dungeonmaster and countless Dens of Iniquity to trap the unwary. The market research people say the sales potential for the product is immense if you can get it on the market for £5.50 and sell at least 65,000 copies. Unfortunately, the appeal of the product is such that the number of Spectrum owners alone would be unlikely to generate the required number of sales, and at least one more version would be needed. The cost of producing new versions for, say, the Oric and the BBC Micro would push the unit cost up to £6.70 and, at this price, the sales would drop below an economically viable level. This is the dilemma that frustrates many prospective software writers.

The problem of software compatibility has not been ignored by the computer industry. The individualistic West does not look kindly on standardisation, but computer developers in the Far East prefer things to be systematic, ordered and standardised, especially when this translates into profits.

ASCII/Microsoft is attempting to change the chaos into order. The company is the result of a merger involving the American Microsoft Corporation and ASCII, a successful Japanese publishing company with a string of popular magazines to its credit. As a side-line, ASCII also publishes commercial software, and when Microsoft wanted to penetrate the 'impenetrable' Japanese market, ASCII seemed a natural company to turn to for a joint venture. Microsoft had the technical expertise and ASCII had the marketing experience.

The American Microsoft Corporation had made its name on the nearest thing BASIC has to a standard, Microsoft MBASIC, which has been adopted by computer makers all over the world. But even so, no MBASIC program could be guaranteed to run on all computers using the language whenever any special features were present, simply because of the lack of hardware compatibility.

Microsoft BASIC was successfully sold to numerous Japanese computer manufacturers through the offices of ASCII/Microsoft. But still this didn't solve the problem of hardware and software compatibility. ASCII/Microsoft's solution was to create a standard, in co-operation with leading Japanese manufacturers, which they hoped would become internationally recognised. What they eventually came up with is called the MSX Standard. The specification includes basic hardware requirements (based around the Z80 microprocessor and certain other chips), together with a standardised language.

MSX SPECIFICATIONS

MSX BASIC is very similar to Microsoft's MBASIC but with a number of significant enhancements to take advantage of today's graphics and sound capabilities. Extra statements include SCREEN, to specify the screen mode, sprite sizes, key 'click', cassette baud rate and printer options; LOCATE for character positioning on the screen; COLOUR for selecting one of 16 foreground and background colours; PUT SPRITE for setting up sprite attributes; CIRCLE for drawing circles and ellipses; DRAW for drawing figures; LINE for drawing lines between specified co-ordinates; and PAINT for filling figures with a specified colour. A KEY statement is also provided for assigning strings to function keys. Further statements provide for poking values to the video RAM (VPOKE), writing values to the registers of the sound effects chip (SOUND) and cassette motor control (MOTOR).

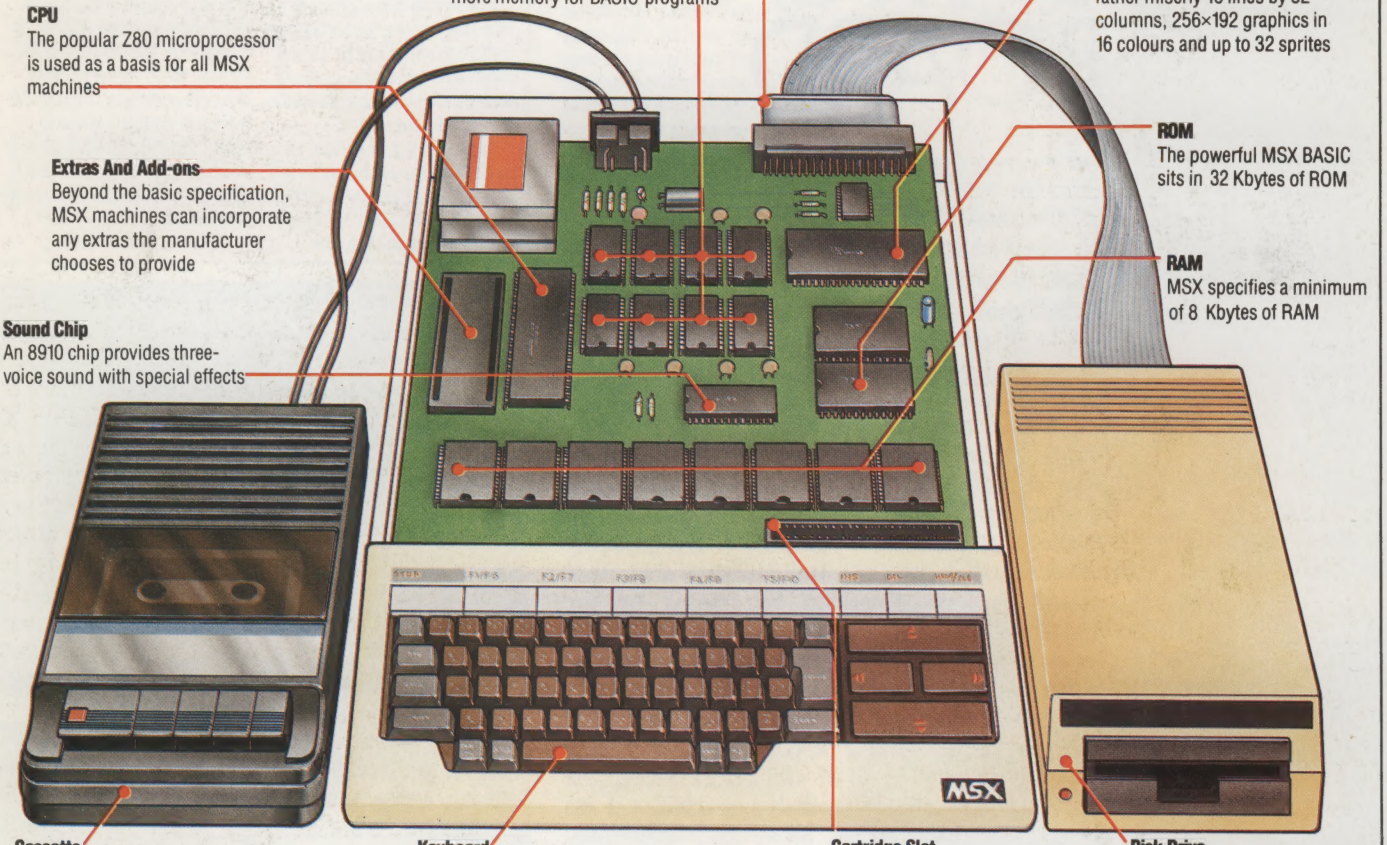
MSX involves more than just standardised software, however. The CPU is specified as a Z80 running at 3.58MHz. There must be at least 32 Kbytes of ROM to store the MSX software. Additionally, there must be at least eight Kbytes of RAM. There is no upper limit on the allowable amount of ROM and RAM. An MSX computer must incorporate a Texas Instruments TMS9918A video controller chip (or equivalent) and an AY-3-8910 — a three-voice sound generator chip. The video output must be capable of displaying either 32 columns by 24 lines, or 40 columns by 24 lines. There is no provision in the standard for an 80 column display at present. A resolution of 256 by 192 pixels is required.

The cassette interface has been established, cassettes being the primary means of storing programs and data. It must use the FSK encoding system at 1,200/2,400 bits per second. The keyboard, reflecting MSX's Japanese pedigree, has not only a standard layout, including function keys, but also provision for Katakana (the Japanese 'angular' alphabet), Hiragana (the 'cursive' alphabet), standardised graphics characters and, optionally, Kanji (the Chinese character writing system).

Plug-in software is catered for through a standard ROM cartridge slot and there is a 50-pin standard I/O bus. There is even a standardised port for two joysticks.

Disk formats are also standardised, as is the disk operating system, MSX-DOS. It is functionally equivalent to MS-DOS and enables MS-DOS data files to be read. It is also said to be compatible with the hugely popular CP/M 2.2 disk operating

MSX Architecture


Video RAM
Some systems can have separate video RAM, thus freeing more memory for BASIC programs
Expansion Connector
All machines have a defined expansion connector
Display Chip
A Texas 9918 or equivalent provides the display with a rather miserly 16 lines by 32 columns, 256×192 graphics in 16 colours and up to 32 sprites
CPU
The popular Z80 microprocessor is used as a basis for all MSX machines
ROM
The powerful MSX BASIC sits in 32 Kbytes of ROM
Extras And Add-ons
Beyond the basic specification, MSX machines can incorporate any extras the manufacturer chooses to provide
RAM
MSX specifies a minimum of 8 Kbytes of RAM
Sound Chip
An 8910 chip provides three-voice sound with special effects
MSX
Cassette
MSX machines record on ordinary cassette recorders at a fast rate of either 1,200 or 2,400 baud
Keyboard
MSX machines all have the same keys, although their exact positioning may vary
Cartridge Slot
A standardised cartridge slot allows any MSX cartridge to be plugged in
Disk Drive
MSX includes a specification for a disk system using a standard format

STEVE CROSS

system. Floppy disk formats for the 3½ inch (Sony), 5¼ inch and 8 inch floppy disks have also been established.

All this means that, if the standards have been adhered to, any program written for one MSX machine, stored on any disk, will be guaranteed to run on any other MSX computer — and be able to take full advantage of its sound and graphics capabilities. The advantages to both manufacturer and consumer are obvious.

But there are disadvantages to the MSX system. The first of these is that any 'standard' will fail to take advantage of innovation in the field. If, for example, a new video controller chip comes along with vastly superior capabilities, MSX programs will not be able to take advantage of it and will leave the field wide open to competitors able to provide software capable of utilising the potential of the new chip.

The second is that eight-bit microprocessors, of which the Z80 is the most successful, have a limited life. Eight-bit processors are inherently unable to address more than 64 Kbytes of main memory directly, nor are they capable of handling data bigger in value than 256 at any one time. In this light, the MSX standard could be seen as a last ditch attempt at prolonging the life of the Z80, which is destined to enjoy only a short-term success in the market place.

MSX may also provide a pointer to the future. It is hard to suppose seriously that the microcomputer market will be dominated by eight-bit processors in five or 10 years time. If the MSX standard achieves anything it is likely to remind computer developers that standardisation should come early, rather than late, in the life of any computer innovation. MSX may make life easier for the big boys with Z80-based products waiting to be sold at the moment; it is difficult to see how it will have any long-term impact beyond convincing the rest of the world that standardisation does matter. In the field of 16-bit computers, IBM has proved that 'might is right' with its personal computer, which has become a de facto standard. Will the MSX standard be able to do the same for the eight-bit home micro? So far MSX is being supported by a total of 16 manufacturers, including Yamaha, JVC, Hitachi, Sony, Sanyo, National, Pioneer, Canon, Fujitsu and Mitsubishi from Japan, the American company Spectravideo and Daewoo from Korea. No British manufacturer has joined the club so far. Only time and the market will show if the world needs more of the same, or the kind of individualistic innovation we have come to expect from entrepreneurs like Sir Clive Sinclair.

Design Convention
To make programs and add-ons compatible with all MSX systems, the design of an MSX micro follows strict rules. Once the machine reaches the basic specification shown here, designers can add their own special extras

SWITCH BOARD

In the last instalment of Workshop we introduced the most important components of electronics: diodes, resistors, capacitors and transistors. In computing, the most significant of these is the transistor. Here, we look at what it does and use it to build some simple logic gates.

Transistors have two essential roles; to act as an amplifier for a signal, or to turn one current on and off under the control of another current. It is this ability to act as an electronic switch that makes them useful in computers. By grouping them together, you can build circuits that store on/off patterns that can be treated by the computer as binary numbers. Other circuits are logic gates that let you add on/off sequences together and so on.

If you build the logic gates described on the page opposite, you'll appreciate that whole computers made out of transistors would be extremely large and expensive machines — as indeed they once were. Small and reasonably priced computers need a further refinement — integrated circuits. These are predefined circuits with hundreds of transistors etched onto a tiny chip of silicon, enclosed in a black plastic case. You can buy simple integrated circuits (known as 'TTL chips') for most jobs. Four AND gates on one chip will cost just a few pence. The chips necessary for complex tasks, such as binary counters, and so on are about 50 pence each.

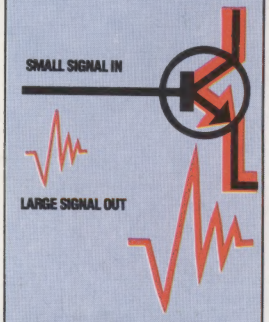
The practice of placing more and more circuits onto a single chip is often termed 'very large scale integration'. VLSI chips have many thousands of components compressed into them and can do very complex jobs. The microprocessor is such a chip. So are the chips that generate the television display, control the interfaces and provide the range of sound effects that many machines can provide. The principles are the same as those we have used for three simple logic gates. It is only the sheer number of components and complexity of the circuits that differ.

VLSI chips are made from a number of different technologies, depending on a trade-off between economy, performance and power consumption. The type found in most computers are MOS chips (metal-oxide-silicon), while many battery powered portables use CMOS chips (complementary metal-oxide-silicon), which are slower but use far less power.

In the next instalment of Workshop we'll build a half adder, based on the circuit described on page 33 of the Computer Science series.

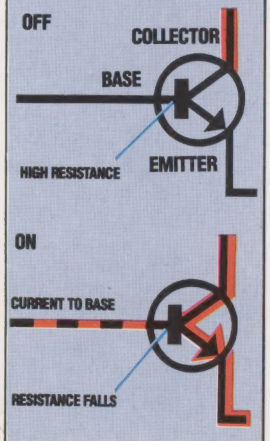
Transistors As Amplifiers

A transistor has three parts, known as a base, collector and emitter. Three wires, each connected to one of these, run out through the casing of the transistor. Typically, the base is used to control a current flowing from the collector to the emitter. If a current is applied to the base, a current can flow from the collector to the emitter. A changing current applied to the base causes the resistance of the collector to emitter path to vary in unison. By feeding a large current into the collector and varying it with a small current through the base, the emitter produces a large signal that varies exactly as the small signal. The transistor has been used to amplify the small signal.



Transistors As Switches

Switching with a transistor involves using the properties of the base, collector and emitter to their limits. A transistor can let only so much current through its collector to emitter path. It has a saturation point at which the amount of current flowing through the collector to emitter path is no longer affected by small variations in the amount of current flowing through the base.

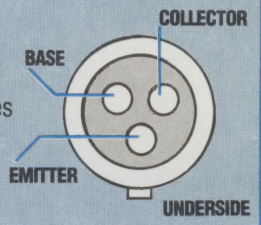


Building The Logic Gates

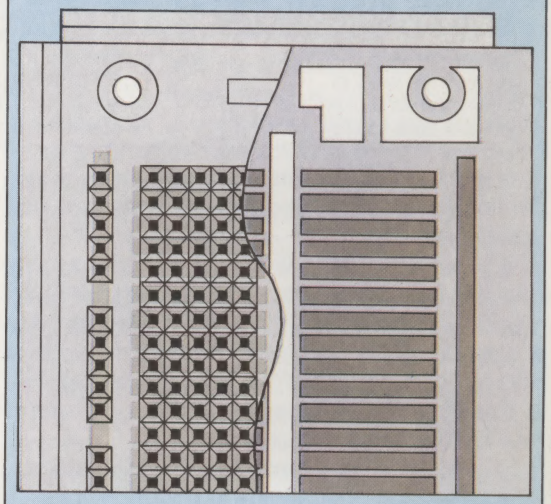
These simple circuits (right) illustrate the way the transistor's switching ability can be used to build logic gates. You can build each one in turn using the same set of components and a breadboard. It is important to realise that the actual switching is 'solid-state', in other words it has no moving parts. In these examples the inputs to the circuits are push buttons with LEDs (light emitting diodes). In a computer, the inputs to such circuits would be the outputs from other circuits: Once you've built and understood these gates, you might even like to try building a more complex circuit by feeding the output of one gate into another.

What You Need

1 Breadboard (Experimentor 300 or similar)
2 BC109 transistors
2 Red LEDs
1 Green LED
3 500 ohm resistors
2 15K ohm resistors
2 Push-to-make switches
1 9 volt battery
1 Battery clip
Short lengths of wire

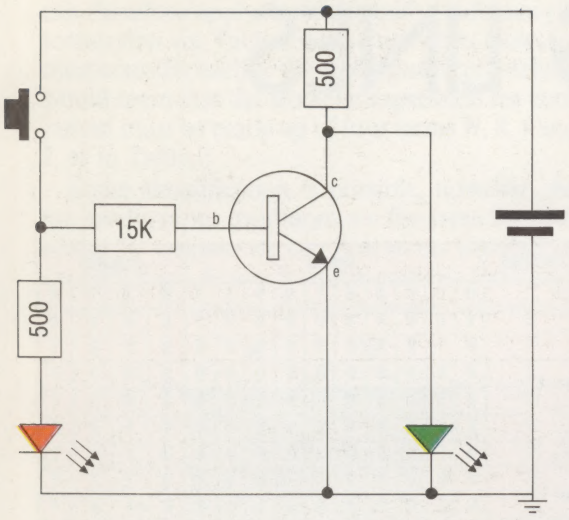


Breadboards



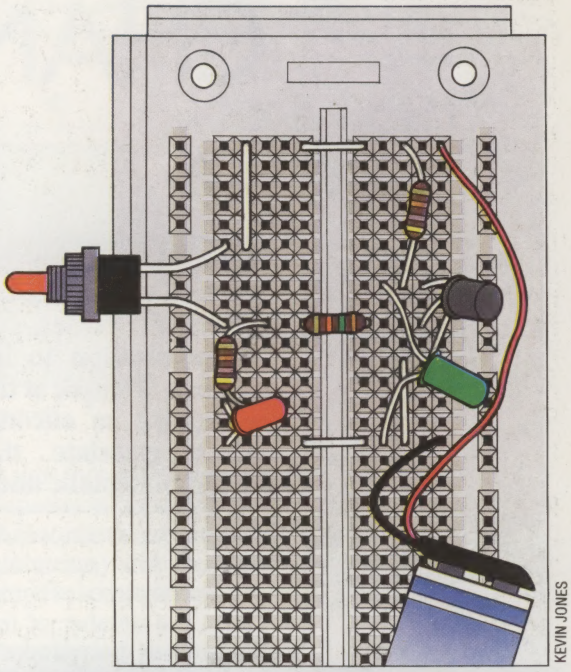
KEVIN JONES

Breadboards provide a simple way to experiment with circuits like these without having to take the time and trouble to solder the components. The breadboard is a re-usable base into which components can be securely plugged. The metal grips for the components act as conductors, so each group of five holes is electrically connected. With this matrix, it's easy to transfer simple circuits onto the board, using short pieces of wire to connect separate groups of holes. The board illustrated here is larger than you need at present but it will be useful for future projects.

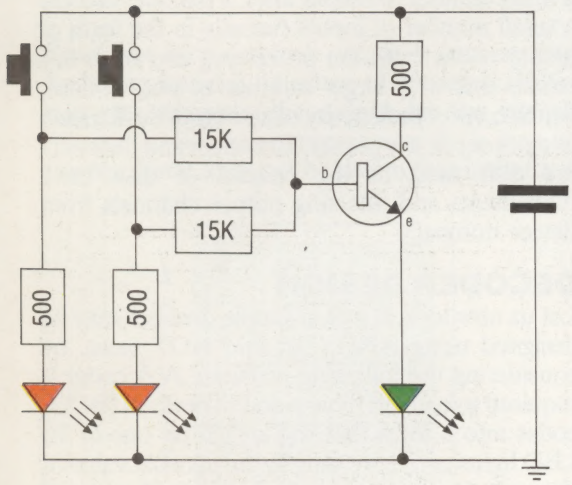


The NOT Gate

This is the simplest of logic gates, with only a single input (indicated by the red LED and switch) and a single output (the green LED). With the switch open, no current can flow through the base to the emitter of the transistor. This makes a high resistance at the collector and as a result the current flow takes the alternative route through the green LED. Therefore, when the button is not pushed (that is, an input of 0), the green LED lights up (an output of 1). Pushing the button feeds a current into the base of the transistor, removing the resistance from the collector. The current now flows through the transistor, avoiding the LED. So when the button is pushed (an input of 1), the LED doesn't light (an output of 0).

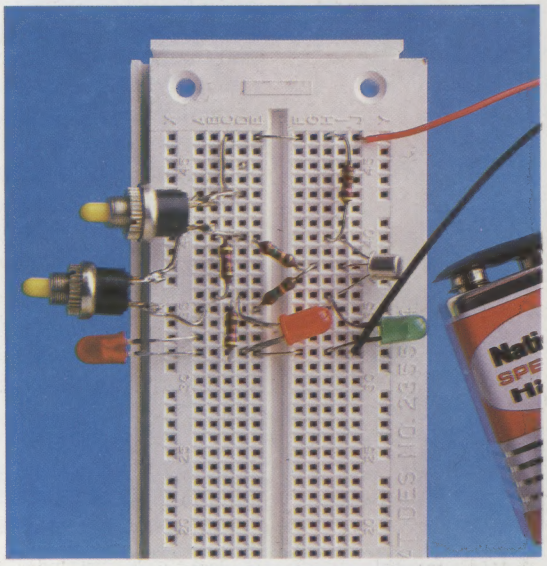


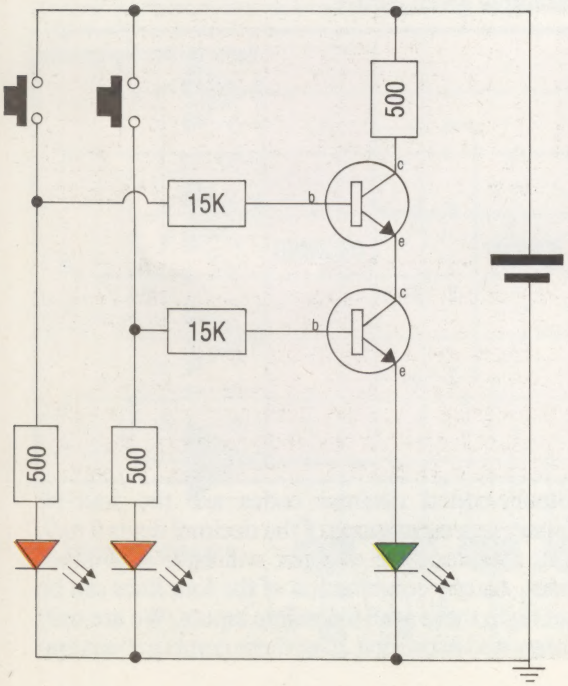
KEVIN JONES



The OR Gate

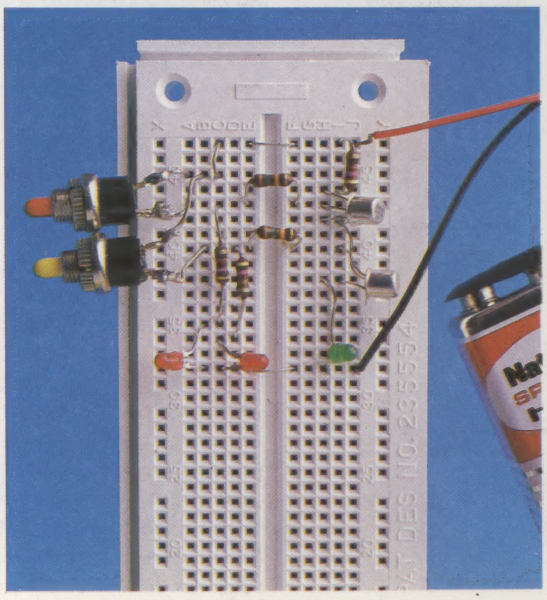
The circuit used for the NOT gate can be simply adapted to become an OR gate. The first change is to place the output LED so that it will be driven by a current *through* the transistor. There are two inputs to an OR gate, each with a switch and LED. When either switch feeds a signal into the base, it switches the transistor to allow current through to the LED.





The AND Gate

To create an AND gate, we need two transistors on the route to the output, each with its own input. Only when both switches are held down (that is, both inputs are 1) will both transistors be open, allowing the current to flow through to the LED.



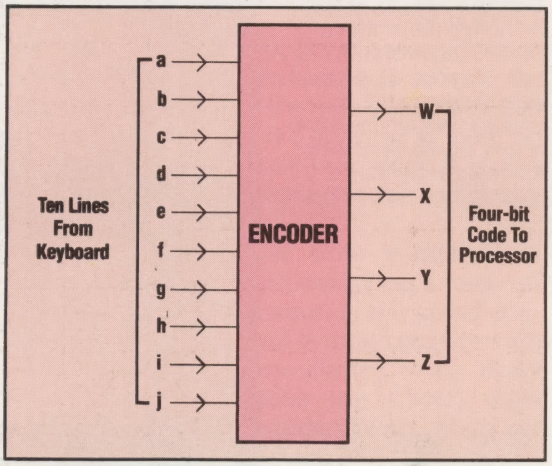
IAN McKINNELL

CHANGING LINES

The CPU of a home computer carries out its work by sending instructions, in the form of electrical pulses, down channels of communication to internal or peripheral devices. We look at the special logic circuits — known as encoders and decoders — which translate the instructions into electrical signals, and vice versa.

The processor sends instructions to both internal devices (the accumulator, ALU, etc.) and peripheral pieces of equipment (such as a printer). Often the number of lines used by a peripheral device may be reduced to provide a simplified input to the processor. This is known as *encoding*. An example of an encoder is a circuit used in conjunction with a keyboard. This may have 64 output lines, one of which produces a signal when the corresponding key is depressed. As only one key is usually pressed at any one time, each of the 64 possible output signals can be coded as a six-bit binary number. This means that only six lines are required to carry the information to the processor about which key was pressed. The device that converts 64 lines to six lines is an encoder. In practice, a further two lines are added, one to be used as a parity check and the other to signal that a Shift or Control key has been used in conjunction with another key.

To demonstrate this principle let's consider a much simpler keyboard that has only 10 keys, which would allow a user to type in a number from 0 to 9 inclusive. A three-bit binary code would give us only eight possible combinations, so we must design our encoder to accept 10 lines of input and produce four lines of output.



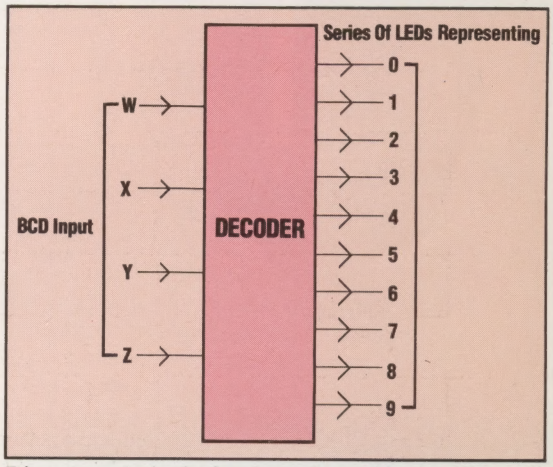
As only one of the 10 lines can be activated (or *set high*) at any one time, the truth table for the encoder will be:

Decimal	Inputs										Outputs			
	a	b	c	d	e	f	g	h	i	j	W	X	Y	Z
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0	0	0	1	0
3	0	0	0	1	0	0	0	0	0	0	0	0	1	1
4	0	0	0	0	1	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	0	0	0	0	1	0	1
6	0	0	0	0	0	0	1	0	0	0	0	1	1	0
7	0	0	0	0	0	0	0	1	0	0	0	1	1	1
8	0	0	0	0	0	0	0	0	1	0	1	0	0	0
9	0	0	0	0	0	0	0	0	0	1	1	0	0	1

Decoding is the reverse of encoding. Rather than accepting a large number of input lines to produce a small number of output lines, a decoder accepts a small number of inputs (usually in the form of binary codes from the processor) and from this selects one of a larger number of output lines, which control the activity of an output device such as a printer or x-y plotter. Encoders and decoders are also used in the control of disk head movements and selecting output channels from device numbers.

DECODER DESIGN

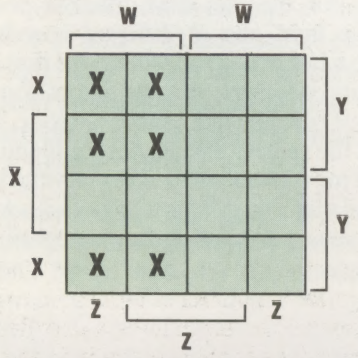
Let us now look at how a simple decoder may be designed using AND, OR and NOT gates, by considering the following problem. A decoder is required to convert binary-coded decimal (BCD) codes into a form that will switch on one of 10 LED lights corresponding to the decimal value of the code. Here we are really dealing with a circuit that will produce the reverse of the encoder example given earlier.



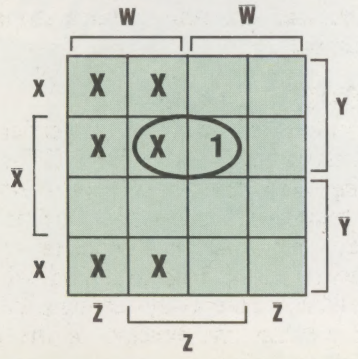
Binary-coded decimal codes are the four-bit binary representations of the decimal digits 0 to 9, and therefore the decoder will have four input lines. As any combination of the four lines can be set high, there are 16 possible inputs. We are only interested in the first 10 of these combinations and

can therefore apply 'invalid input' conditions (X) to the other six. Table 1 is the truth table. Now, we must consider each of the 10 outputs separately. It would seem that the Boolean expression for each output must be made up of four terms W, X, Y and Z, as in Table 2.

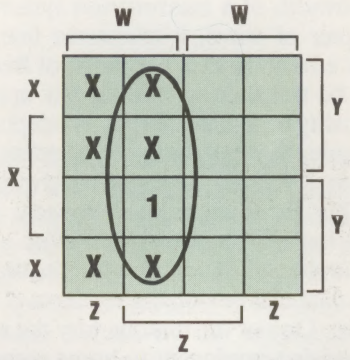
Some simplification is possible, however. As the invalid input conditions are the same for each of the 10 outputs we can link terms together to simplify them. Below is a four-variable Karnaugh map with the six invalid input conditions marked.



If we now consider each output in turn we may find that some simplification can be made. Let us take, for example, the output for the digit 3. Placing the Boolean expression on the k-map we can see that a loop can be drawn that includes it.



Thus the expression for the output can be simplified to $\bar{X}.Y.Z$. Taking output 9 as a second example:



A loop can be drawn that uses three of the invalid input conditions and represents the Boolean expression W.Z. Many of the other output terms simplify in a similar way. You may wish to check for yourselves that the simplified output terms are those shown in Table 3.

All that now remains to be done is to construct the circuit diagram from the 10 Boolean expressions. As each input is used both in its normal form and its negative form it is easiest to construct the circuit from eight parallel lines representing these terms. Each of the 10 outputs can then be formed by branching off from the relevant lines and using AND gates. The completed decoder circuit diagram is given below.

Exercise 6

1) A three-input encoder is to be designed to create an output of 1 for inputs 011, 101, 110, or 111. The output should be zero for all other input combinations.

a) Draw the truth table for the encoder

b) Produce a Boolean expression for the output and simplify it

c) Draw the encoder circuit

Table 1

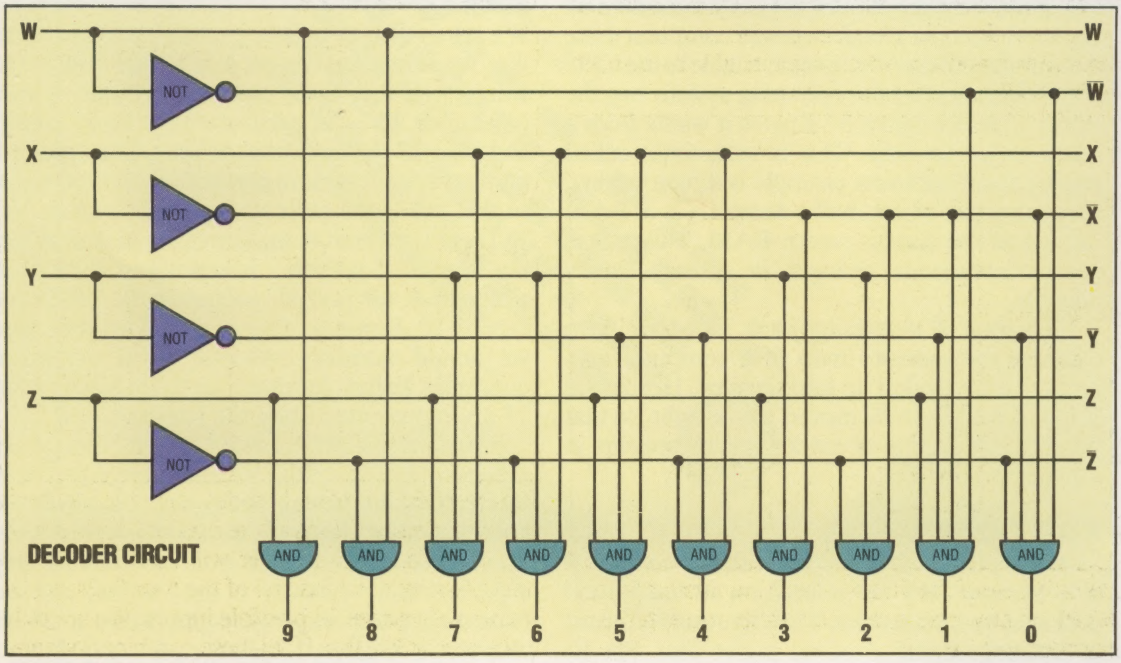
Inputs				BCD Digit
W	X	Y	Z	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Table 2

BCD Digit	Boolean Expression
0	$\bar{W}.\bar{X}.\bar{Y}.\bar{Z}$
1	$\bar{W}.\bar{X}.\bar{Y}.Z$
2	$\bar{W}.\bar{X}.Y.\bar{Z}$
3	$\bar{W}.\bar{X}.Y.Z$
4	$\bar{W}.X.\bar{Y}.\bar{Z}$
5	$\bar{W}.X.\bar{Y}.Z$
6	$\bar{W}.X.Y.\bar{Z}$
7	$\bar{W}.X.Y.Z$
8	$W.\bar{X}.\bar{Y}.\bar{Z}$
9	$W.\bar{X}.\bar{Y}.Z$

Table 3

BCD Digit	Boolean Expression
0	$\bar{W}.\bar{X}.\bar{Y}.\bar{Z}$
1	$\bar{W}.\bar{X}.\bar{Y}.Z$
2	$\bar{X}.Y.\bar{Z}$
3	$\bar{X}.Y.Z$
4	$X.\bar{Y}.\bar{Z}$
5	$X.\bar{Y}.Z$
6	$X.Y.\bar{Z}$
7	$X.Y.Z$
8	$W.\bar{Z}$
9	$W.Z$



LIZ DIXON

B

BANDWIDTH

The term *bandwidth* is used in connection with the communication of information over a distance. The bandwidth of a *transmission system* (be it a twisted pair of wires, a telephone line, a laser beam, or a radio link) is the range of frequencies that can be transmitted without any appreciable loss in quality or signal strength. A telephone line, for example, can handle audio frequencies from 300 Hz to 3,400 Hz, giving a bandwidth of 3.1 KHz. This is adequate for speech, but not sufficient for hi-fi music reproduction.

The bandwidth determines the maximum rate at which data can be transmitted from one device to another. One of the reasons why the advent of cable television is relevant to home computing is that the cables need to be of an extremely high bandwidth, as one television channel occupies as much bandwidth as around 3,000 telephone conversations. Low-cost modems generally can't transmit data faster than about 1,200 baud using telephone lines. But linked to a cable television network, it would be possible to send an entire program to another user in a fraction of a second.

BANK SWITCHING

Every microprocessor or CPU has a fixed address range, determining the maximum number of memory locations that it can access individually. On an eight-bit CPU, such as the Z80 or 6502 found at the heart of most home computers, this address range corresponds to 64 Kbytes (i.e. locations 0 to 65,535), and this must accommodate all the RAM, ROM and input/output chips needed by the system. The newer, 16-bit business systems (including the Sinclair QL) can address much more memory — in some cases several Megabytes.

The only way in which you can add more than 64 Kbytes of memory to an eight-bit device is to use *bank switching*. This technique is used as standard on machines such as the Commodore 64 and the Lynx. On a bank switched computer there is never more than 64 Kbytes available to the CPU at any instant, but the operating system has the ability to select which 64 Kbytes it wants from a much larger amount. This is sometimes called *paging* and a common example is screen paging, where any one of, say, half a dozen areas of RAM can act as the current screen RAM. This makes near-instantaneous changes in screen display possible.

One way of visualising bank switching is to consider the memory map of a computer as a vertical strip divided up into sections. Horizontal strips of RAM can be moved left or right, so that the required section of extra memory appears in the memory map.

BAR CODES

Most people are now familiar with *bar codes* — if only because they are printed on so many food packages and magazines. They represent a means of printing digital data on paper that can be conveniently input to an electronic device. All the user has to do is pass a light pen across the printed strip of lines, which encode the data.

The use of bar codes is not restricted to the world of supermarkets, however. They have also been used as a means of entering music into a digital synthesiser (on the Casio organs) and Hewlett Packard pioneered the use of bar codes for entering programs from paper into a microcomputer.

The main difficulty in reading bar codes is in allowing for the fact that different people will move the light pen at different speeds. If you examine a bar code, you will notice that there are three pairs of synchronisation marks (long double lines): one pair at either end of the code, as well as a pair in the centre. The process of identifying the speed of movement and then retrieving the data is very similar to that of extracting the clock-pulse in synchronous transmission (see page 108) over the telephone.

IAN McKINNELL

Storing Chords
Bar codes aren't just used for price tags on goods. This Casiotone organ stores music as bar codes to be read into its programmable memory

BASE

We are so used to counting in the decimal system that we sometimes forget it is a base system. In computing, it is more convenient to use a base other than 10 — in particular base two (binary) and base 16 (hexadecimal or hex). The base of a number system is more correctly called the *radix*, so that two is the radix of the binary system.

There is no limit to the size of the base or radix: the system of counting minutes and seconds is really base 60 (called *sexagesimal*). When we exceed 10, however, we run out of symbols that we would normally associate with numerical quantities. In hex, for example, the numbers 10 to 15 are represented using the letters A to F.

The choice of which base to operate in is purely arbitrary. The use of the decimal system probably results from our having 10 fingers; computers use binary because electronic devices have two stable states. The choice of hex is more involved: any eight-bit value (usually a data value) can be expressed using only two digits, whilst a 16-bit number (a memory address) needs only four.

VIDEO STAR

The American Spectravideo computers incorporate many of the features of the MSX standard devised by a number of Japanese and American companies. Although they are newcomers to the European computer market, their low price and potential for expansion should assure them a bright future.

The Spectravideo 318 is a low-cost home computer that is undeniably of the same quality as the BBC Micro and the Commodore 64. It incorporates a three-voice sound synthesiser, high resolution graphics with sprites, a built-in joystick and a cartridge port.

The keyboard is very close to the MSX specifications (see page 141). It is of similar design to that of the Sinclair Spectrum and has the same style rubber pressure pad keys. However, the keys are well spaced out and the keyboard is generally much more pleasant to use.

The cursor pad has been turned into a built-in joystick. A disc replaces the usual four arrow keys and you can tap the disc in the relevant place to move up, down, left or right. Alternatively, you can slot in a joystick handle and generate the key presses simply by moving the joystick. You can also use the joystick to position the cursor over mistakes while correcting a program.

The Spectravideo's BASIC is the latest in a line of interpreters written by the Microsoft company. This version is close to MSX BASIC, itself an extension of the GW BASIC used on machines such as the IBM PC. BASIC programs are easily developed with the full screen editor and general facilities such as renumbering and automatic line numbering. Although there are none of the new 'structured' commands found in many new BASICS such as WHILE...WEND and REPEAT...UNTIL, this version does have IF...THEN...ELSE, which is necessary for writing neat and efficient programs. The BASIC also takes full advantage of the Spectravideo's graphics.

The screen has a reasonable resolution of 256 by 192 dots in 16 colours, although small groups of dots must share the same colours. This may seem poor compared to machines such as the BBC Micro, but it is the level of control over the screen that makes for good graphics and not

IAN McKINNELL

Spectravideo Keyboard
The 318's keyboard is very close to the MSX specifications. It is a Spectrum-style rubber mat design but the spacing and quality of the keys make it perfectly usable. It is a copiously equipped keyboard with all the usual keys such as Control, Escape, Tab and Backspace, plus five function keys (each with two functions), a STOP key, a SELECT key and a set of editing keys such as INS, DEL and COPY. A set of graphics shapes, such as blocks, dots and lines is available by pressing a letter key in combination with either the LEFT GRPH key or the RIGHT GRPH key. The shapes produced by each key are clearly marked on the keyboard

merely its specification. The 318 has all the Microsoft BASIC graphics extensions, for example single commands to draw points, lines, boxes, circles, arcs and ellipses. A PAINT command fills any enclosed shape with a particular colour. If you want greater control, special VPOKE and VPEEK instructions write to and read from the screen memory directly. There's also a mini graphics language used with the DRAW command to create complex shapes and drawings.

Any rectangle of the screen can be 'picked up' by reading it off the screen into a BASIC array with GET and replaced on the screen with PUT. These commands make it easy to produce regular patterns, simple animation or special effects such as reversing out an image. The finishing touch is that the 318 has sprite graphics. The 9929 display chip provides the facility for programmers to design their own animated shapes, such as people, space invaders, or missiles.

The ON SPRITE GOSUB instruction enables you to set up an 'event trap'. In this case, the program proceeds quite normally, but if two sprites collide, BASIC will jump to a special routine to deal with the collision. This could be used to detect a missile hitting a spacecraft for example. Using this interrupt facility, the programmer has no need to check continually for all the events that might happen. As a result, the programming is easier and the program runs much quicker. Similar event trapping is available for the function keys, the cursor keys and so on.

Good sound is an essential requirement and the Spectravideo's sound chip has three voices and a range of special effects. This can produce some impressive results, although the more complex effects are quite difficult to achieve from BASIC. The sound is replayed through the television, providing a convenient volume control.

The Spectravideo has a small selection of interfaces: two joystick ports, a cartridge port, a cassette port and an expansion connector. There is an interesting range of add-ons available, but they tend to be rather expensive. To connect any add-on, you need to buy the mini-expander. This will allow you to add one extra option, usually a 16 or 64 Kbyte memory expansion. Further expansion is via the super-expander that will provide for up to seven add-ons, using a slot system similar to that of the Apple II. You can plug in more memory here, as well as printer interfaces, disk drives and modems. If you are a games enthusiast, you could opt for a very interesting add-on, the Coleco games adaptor. This will enable the 318 to run cartridges from a Coleco television game, although it is a comparatively expensive way of running games software.

IAN McKINNELL

Spectravideo Joystick
A built-in joystick replaces the four arrow keys you'll find on most computers. Without the handle, you can tap the disc in the right spot to move up, down, left or right. With the handle in position, moving the joystick tips the disc in the appropriate direction. This is more than a cosmetic improvement as the joystick allows you to move diagonally by pushing in two directions at once; something that would be hard to achieve using four separate keys

Cartridge Port
Cartridges plug in through the top of the case into a firmly mounted socket

ROM
Spectravideo BASIC resides in two 16 Kbyte ROMs

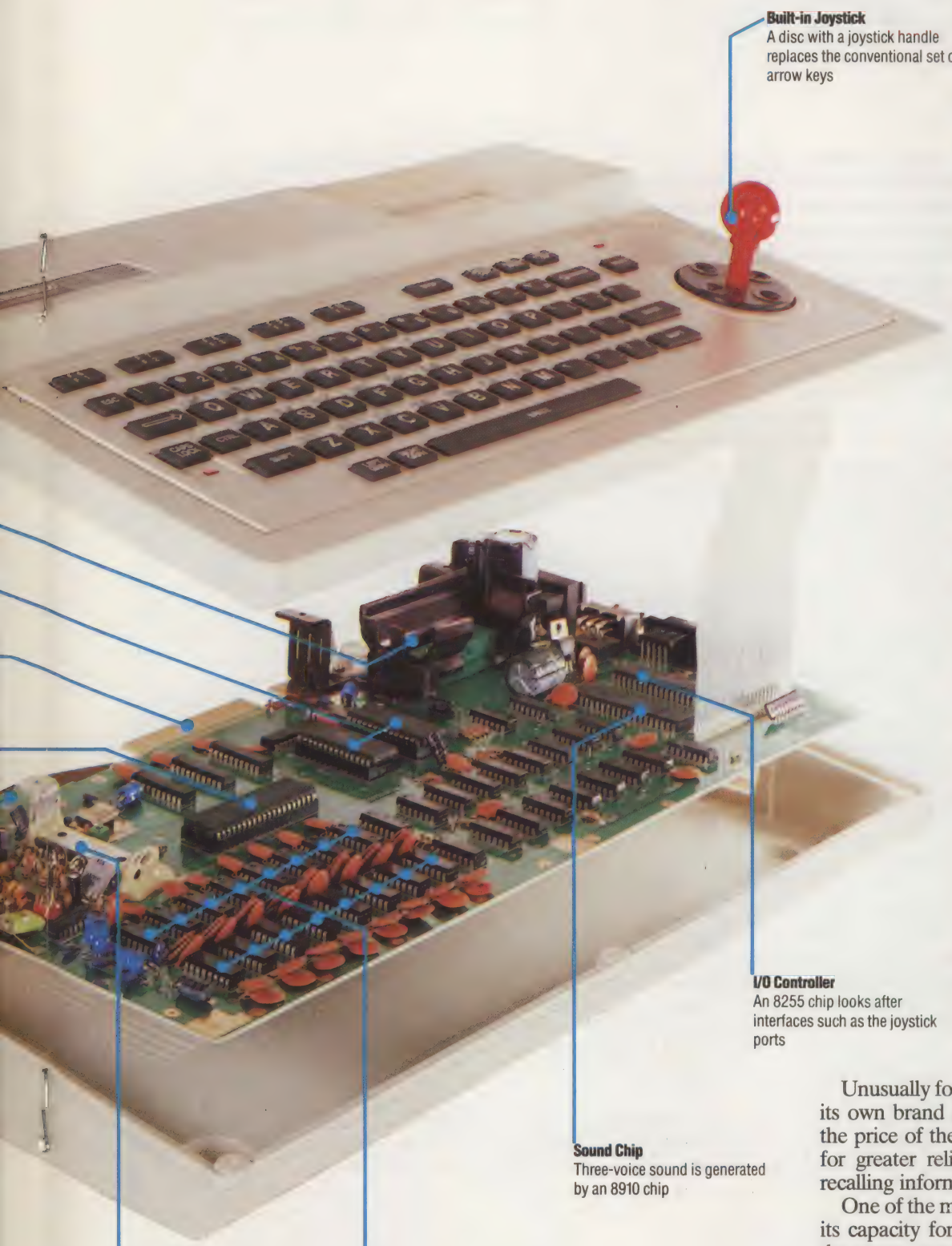
Expansion Connector
A range of expansion boxes and options connect here

CPU
The popular Z80 is used as the processor chip

Monitor Output
A separate modulator connects here to drive a television set. This enables the Spectravideo to be used with different standard televisions by selection of the appropriate modulator

Cassette Port
An edge connector is used as the interface for the Spectravideo's dedicated cassette recorder

Big Brother
An attractive alternative to the 318 is the Spectravideo 328. This is a more sophisticated version, with a full moving keyboard, 80 Kbytes of RAM and built-in word processing software. It costs about £275, and is intended to be a better starting point for users who intend to upgrade to a full business system



Built-in Joystick
A disc with a joystick handle replaces the conventional set of arrow keys

I/O Controller
An 8255 chip looks after interfaces such as the joystick ports

Sound Chip
Three-voice sound is generated by an 8910 chip

Display Chip
A hastily-added heat sink conceals a 9929 display chip. Such 'kludges' are often necessary on early versions of machines

RAM
16 RAM chips provide 32 Kbytes of RAM

SPECTRAVIDEO 318

PRICE
Approx £200

DIMENSIONS
410×220×80 mm

CPU
Z80

MEMORY
32K of RAM, of which around 12K is available for BASIC programs.
32 Kbytes ROM

SCREEN
24 rows of 40 columns text, with 256×192 high resolution graphics in 16 colours and with sprite graphics facilities. An 80-column option is available for business work

INTERFACES
Expansion connector, cartridge port, joystick ports (2)

LANGUAGES AVAILABLE
BASIC

KEYBOARD
Rubber pressure pad, with function and editing keys and built-in joystick/cursor pad

DOCUMENTATION
Skimpy and prone to error, but the arrival of other MSX machines should encourage independent publications.

STRENGTHS
MSX features including an excellent BASIC, sprite graphics, a full keyboard and built-in joystick. Expandability, especially the ability to grow into a full CP/M business computer.

WEAKNESSES
Shortages of software. No printer interface as standard. Needs dedicated cassette recorder

Unusually for a new computer, the 318 requires its own brand cassette recorder. This pushes up the price of the system, although it should make for greater reliability and speed in storing and recalling information.

One of the most attractive features of the 318 is its capacity for expansion. With the addition of the super-expander, 64 Kbytes of RAM, the 80-column card and a disk drive, you have a small CP/M business computer, with access to professional business software. However, it is difficult finding programs that are compatible with the Spectravideo's disk format. This will remain a problem while the machine is new and relatively unknown, but should diminish if the machine becomes popular.

COMPANY REPORTS

We have looked at three different package program approaches to providing a book-keeping solution for small businesses. In the last instalment we compared the pre-structured general ledger with the free format type. We continue our comparative look at three accounting packages, concentrating on their report facilities.

The three packages we are examining are: Cash Trader from Quick Count, Microledger from Lewis Ashley, and Accountant from Compact Accounting Services. As well as recording data about the buying and selling of goods and services, computerised book-keeping packages should be able to deliver a wide range of reports. These can be split into two types. Management reports are intended for internal use by the business. Financial, tax, or VAT reports are designed to be seen by organisations or interested groups outside the business.

As with all computer systems though, the basic data has to be in the system before you can get a report out on it. Cash Trader and Accountant, for example, which do not have the facility to keep master files of supplier and client accounts, cannot produce the range of reports normally associated with sales and purchase ledger systems. Microledger, which has a sales and purchase ledger master file, can. All three packages will provide a printed record, known as an *audit trail,* since it provides proof of all data entries.

Microledger keeps a full transaction history for the current period for each supplier and client account. It can also produce invoices to be sent out to clients showing the balance owed, and the transactions that make up the balance. And it can produce remittance advice notes to be sent to suppliers (showing what they are being paid for). Microledger can also provide, through its sales and purchase ledgers, an index of account names both in numerical and in alphabetical order. Through its nominal ledger, Microledger can provide a full accounts list, a trial balance, and an analysis print-out by nominal code or analysis codes, as well as nominal accounts.

Among the print-out options available on Cash Trader are a summary of the week's takings, a listing of all nominal accounts, and a trial balance. It can also produce statements of cash or bank transactions, summaries of input or output VAT for the quarter, broken down according to VAT code, and final accounts.

Accountant offers nominal ledger reports such as a transaction listing, an account enquiry (listing any particular account) and a trial balance. Sales day book reports include a batch audit trail report, showing all transactions posted for each type of batch posting; a summary batch report, showing postings to the nominal ledger and the VAT control account; and a VAT batch report, showing goods and VAT broken down into VAT category. Purchase day book reports are similar, showing input VAT rather than output VAT.

The aged debtors or creditors report is a good example of an essential management report that is very difficult to assemble by hand. Yet on a computerised system, with a reasonably sophisticated program, it is produced automatically without any additional work from the user. The report shows amounts owed or outstanding, analysed according to whether they are 30, 60 or 90 days overdue. The program reads the date recorded against individual purchases or sales, and sorts them in date order, into one of the three categories. When the amount is recorded as paid, it is automatically erased from the 'aged' analysis.

The trial balance report is common to all three systems. This prints out the account descriptions and the current debit or credit total for each account. Since all three packages use the double entry conventions, the total debits and credits in the system must always balance to zero.

From the trial balance a business will go on to draw up its profit and loss report and its balance sheet. Cash Trader simplifies its profit and loss report by totalling the incomings and outgoings, and then subtracts one from the other. This might be enough for small traders, but it is not a fully comprehensive profit and loss report. Neither of the other two packages attempts to produce either of these reports. They take the user up to the trial balance stage only.

Not all analytical reports can be produced with as little effort from the user as the trial balance and the aged debtors/creditors listing. These, like a couple of the other reports available (such as the transaction listings), are simple 'file dumps' where everything in the file is given some rudimentary formatting plus a heading, and then routed to the printer. Other reports are much more selective, and setting up the criteria for data selection can be tricky, whatever the package you are using.

For example, both Accountant and Microledger have sophisticated analysis and budgeting facilities. But the user has to do quite a bit of work in order to reap the full benefits of the package's reporting capabilities.

Accountant allows the user to set up group keys (or codes) to analyse nominal accounts. If, for

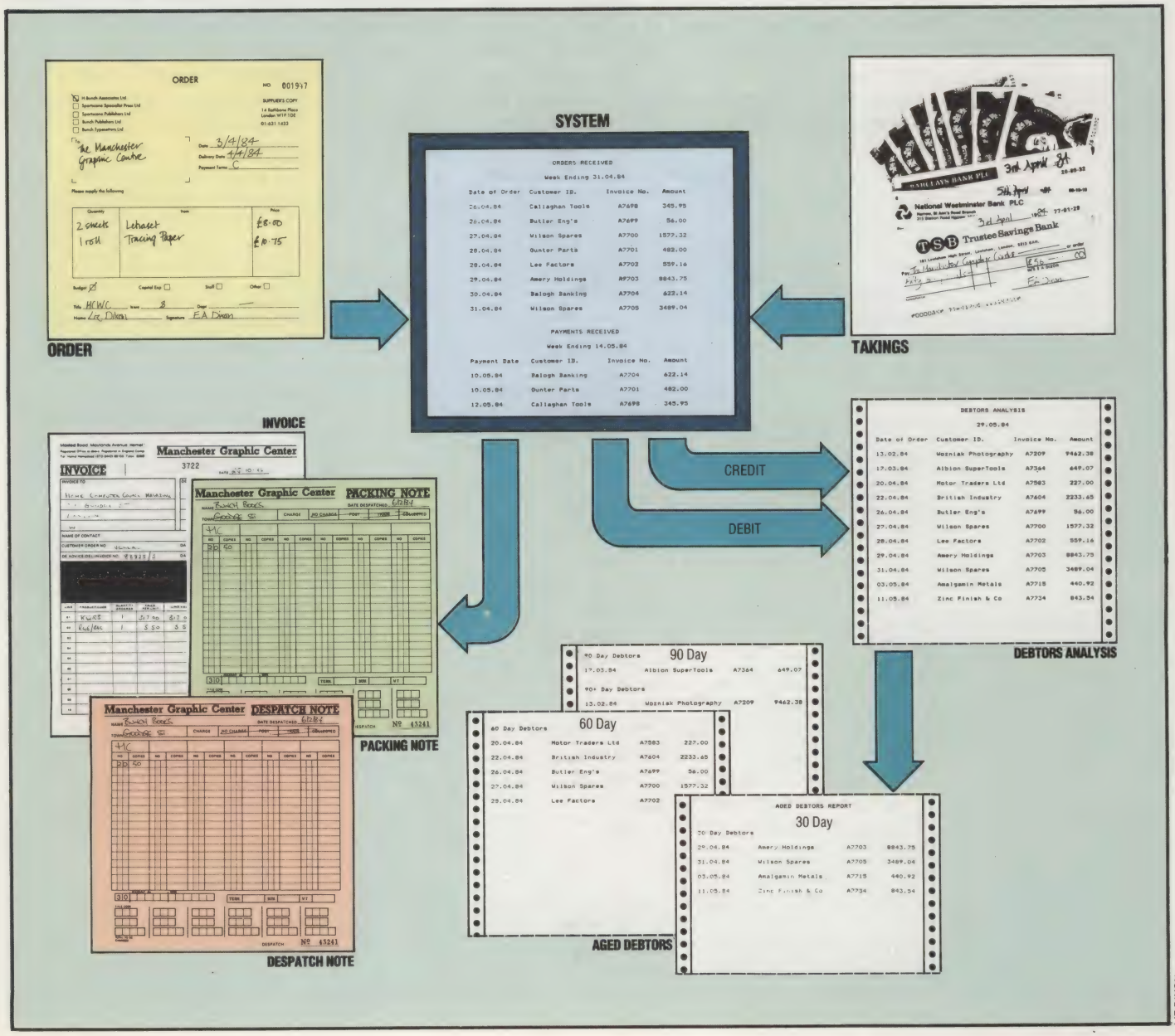
example, you ran a business that consisted of three departments (say production, sales and stores/ purchasing departments), you can use the group keys to find out how much each department cost to maintain. There will be common expenses (and so common nominal account headings) for all three departments. But by giving the purchase department expenses the key 01, the production department 02 and the sales department 03, you can identify what proportion of each expense heading should be allocated to each department. (Production, for example, might account for four fifths of your energy bills.)

Using the 'formatted-trial balance' report option on Accountant, a trial balance can be produced that will group together all accounts by department code, with separate subtotals for each department. This is the sort of management report that can be valuable for internal decision making.

Microledger has a similar facility, only this time the analysis number is built into the account code. Remember, each account, whether sales, purchase or nominal, is identified in Microledger by a three-digit number. There are two analysis levels. The first is a two-digit number prefixed to the account number. The second is a three-digit number sandwiched betweeen the first analysis number and the account code proper (for example: aa/bbb/111).

To produce management reports, this two-level analysis number could be used as follows: let's say there is only one income account on the nominal ledger, but the business has four salesmen. The first analysis number could be used to distinguish the income account from other categories of account, while the second three digits would be more than enough to identify the contributions made by the four salesmen. A report that indicated the individual earnings of the four salesmen could identify those who were performing below average, and be of considerable use to management.

Instant Report
A computerised ledger offers almost instant access to a wide range of useful reports. Here the system compiles information on all transactions and automatically produces an aged debtors report, listing clients who still owe money. A more sophisticated system would use pre-printed stationery to generate its own invoices, despatch and packing notes



A DIFFERENT ANGLE

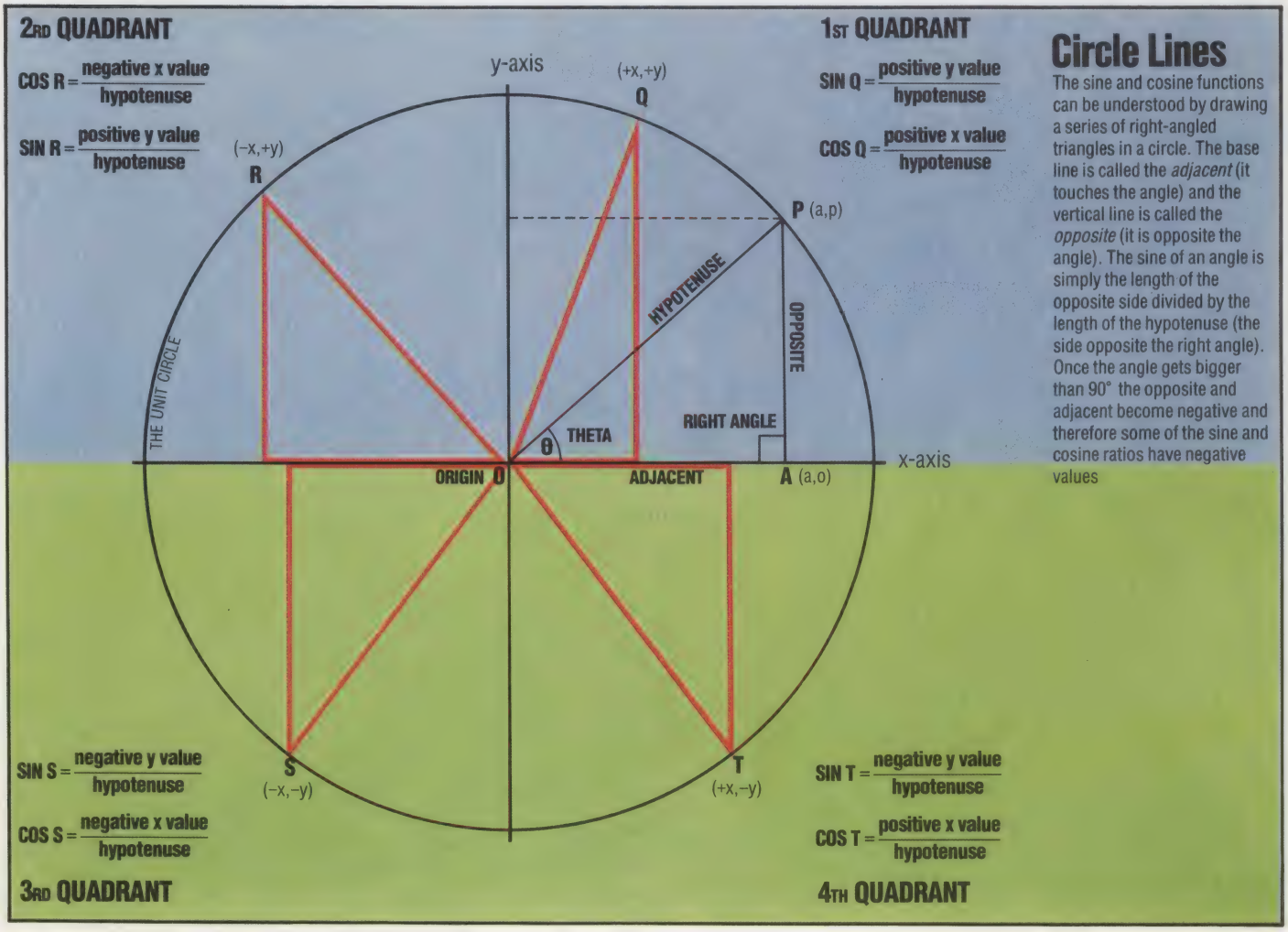
Mathematics is a subject that computer buffs often fight shy of. Nevertheless, there are times when programmers have to use mathematics in their programs in order to get them to work, like it or not! If you love programming, but have grim memories of O Levels and stuffy maths masters, this short series of articles is for you.

What kind of mathematics do programmers actually need to use? That depends, not surprisingly, on the type of program they are writing. The versions of BASIC supplied on many home computers have numerous built-in statements and functions for handling screen graphics — PLOT, CIRCLE, FILL, LINE, COLOUR, INK, PAPER etc. — so the problems of translating and rotating simple figures on the screen are not very great. Even so, there are times when trigonometrical functions such as COS, SIN and TAN will be needed, and, happily, BASIC is furnished with a good set of these. If the terms mean next to nothing to you, don't worry — all will be explained soon.

When it comes to statistics, however, BASIC really lets us down. Most versions of the language have no built-in statistical functions to help with the manipulation of data. If your program needs to predict who will win the next election or whether blue-eyed children will get firsts in their degree exams, you will have to program the functions yourself.

If you write games or typing tutors where the response times of the program user are important, again most versions of BASIC let you down. They simply do not provide the programmer with reliable timing functions. These are the three main areas — trigonometry, statistics and timing — that we will be looking at in this series of Basic Mathematics articles.

High school students of mathematics often



Circle Lines

The sine and cosine functions can be understood by drawing a series of right-angled triangles in a circle. The base line is called the *adjacent* (it touches the angle) and the vertical line is called the *opposite* (it is opposite the angle). The sine of an angle is simply the length of the opposite side divided by the length of the hypotenuse (the side opposite the right angle). Once the angle gets bigger than 90° the opposite and adjacent become negative and therefore some of the sine and cosine ratios have negative values

KEVIN JONES

wonder what relevance to the real world trigonometry could possibly have. The answer lies in the fact that 'trig' (as it is called) is the link between Euclidean geometry, which deals with the manipulation of points, lines and curves, and algebra, which allows mathematical solutions to be arrived at by manipulating variables with unknown values. Take the parabola, for example. This is a curve with many useful properties: and all sorts of things can be discovered about these properties using graph paper, a protractor, a ruler and a pencil. Far more useful, though, is the realisation that the curve can be represented by the algebraic formula: $y = x^2$

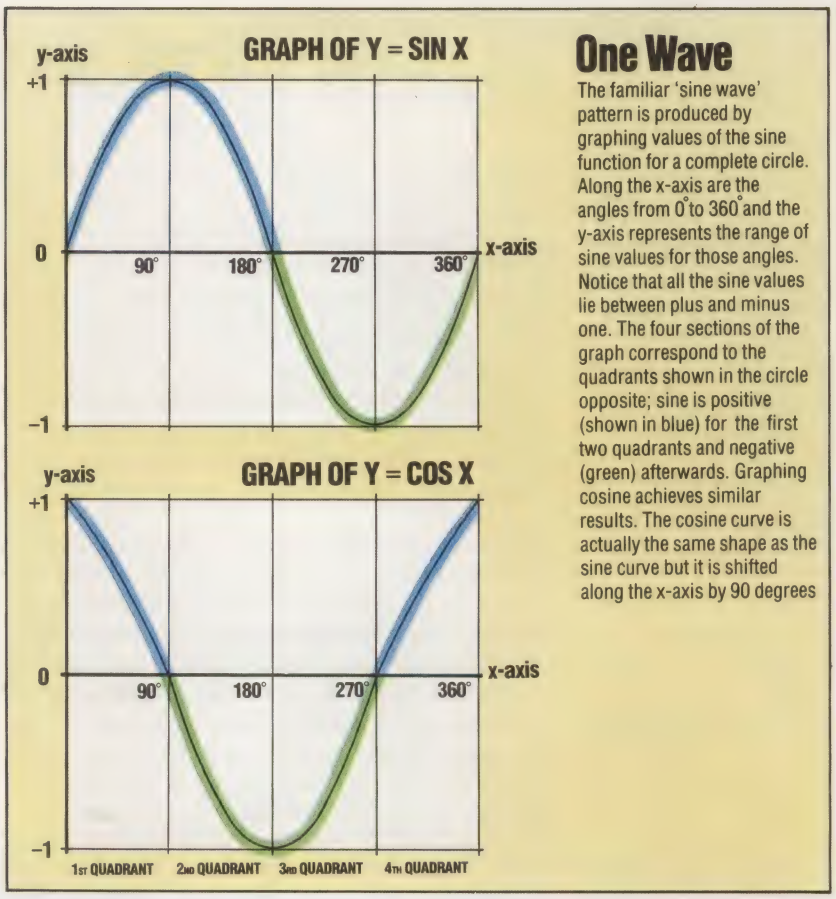
This simple formula allows us to calculate values for any point on the curve without resorting to actually drawing it. Problems that can be dealt with algebraically lend themselves to computer solutions far more easily than problems that require drawn graphs or figures, so the advantage of trigonometrical solutions should be immediately apparent.

The objective of the first two articles of this series is to give a brief overview of trigonometric concepts and to see how these can be quite easily coded into BASIC programs. Let's start by taking a look at the basic trigonometrical functions of *cosine* and *sine*.

CALCULATING THE RATIO

Cosine and sine are two ways of relating the ratio of the sides of right-angled triangles. Cosine and sine are also directly related to the circle. Any right-angled triangle can be drawn to fit inside a circle, called the 'unit' circle. It is given this name because it has a radius of one 'unit' — the actual measurement doesn't matter because it's the ratios that count. The illustration shows a line that has been 'rotated' by 35°. The starting position for a rotation, by convention, is the horizontal axis, and the direction of rotation is anti-clockwise. The horizontal axis is called the *x-axis* and the angle of rotation is called *theta*, written using the Greek letter Θ. If a line is dropped from the circumference to the x-axis, you get a right-angled triangle.

The cosine of Θ is defined as the ratio of the length of the *adjacent* side of the triangle (the part lying along the x-axis) to the *hypotenuse* (the radius of the unit circle). If we take, as an example, a circle with a radius of 15mm, and measure precisely the adjacent side of the triangle inscribed within it, we would get a result of approximately 12.28728mm. Dividing this value by 15 gives a result of 0.819152, which is the cosine value for 35°. If you have a calculator with a COS key (you can also use a set of cosine tables if you know how), try entering 35 COS. You should get a result of 0.819152044. This ratio holds true whatever size of the unit circle the right-angled triangle is inscribed in. Whether the radius of the circle is one inch, one mile or one light year, the side of the triangle lying along the x-axis will always be approximately 0.82 of the radius' value.



One Wave

The familiar 'sine wave' pattern is produced by graphing values of the sine function for a complete circle. Along the x-axis are the angles from 0° to 360° and the y-axis represents the range of sine values for those angles. Notice that all the sine values lie between plus and minus one. The four sections of the graph correspond to the quadrants shown in the circle opposite; sine is positive (shown in blue) for the first two quadrants and negative (green) afterwards. Graphing cosine achieves similar results. The cosine curve is actually the same shape as the sine curve but it is shifted along the x-axis by 90 degrees

We could also draw in other values of Θ in the unit circle illustrated. It can be seen that for any value of Θ the value of cos Θ (cos is the usual abbreviation for cosine, and is pronounced 'koz') will never be greater than 1 nor less than 0. For values of Θ greater than 90°, cos Θ will, however, have a negative value. This is because cos Θ involves the value of the x co-ordinate (the corresponding position on the x-axis of the point where the hypotenuse intersects the circumference of the unit circle). Mathematical convention assumes that the point of origin of the rotation is at 0 on the x-axis. Any points to the left of that will have negative values. For the same reason, the cosine of angles between 180° and 270° will also be negative, but the cosine of angles greater than 270° and up to 360° will again become positive.

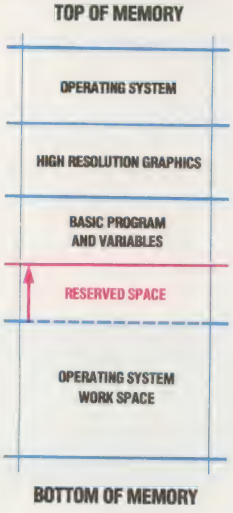
The *sine* function of an angle is very similar to the cosine except that it involves values on the y-axis (the vertical axis). If Θ is 0, the adjacent side of the triangle will be equal in length to the hypotenuse. The co-ordinate of P on the x-axis will always be 1 (because 1/1 = 1), but the co-ordinate on the y-axis will be 0. For all values of Θ up to 90°, sin Θ ('sin', pronounced 'sign', is the usual abbreviation of 'sine') will range from 0 to 1. In the second quadrant of the circle, sin Θ will also be positive, but will decrease from a maximum of 1 down to 0 as Θ approaches 180°. All values of Θ greater than 180° and up to (but not including) 360° will be negative.

STARTING ORDERS

Where To Locate Machine Code Instructions

BBC Micro

In direct mode enter:
PRINT~PAGE
this gives the hex address of the start of your reserved space. Then enter:
PAGE=PAGE+N
where N is the decimal number of bytes you wish to reserve



When a program has been written in its Assembly language form, the machine code programmer must provide directives for the assembler at the beginning of the assembly. We look at several of these 'assembler directives' to see what functions they perform. These instructions may be used with both processors.

In the last instalment of the course, we wrote a simple machine code program that added two numbers into the accumulator and stored the result in memory. There was nothing very startling about what the program did, but, in writing it, we covered many points of significance to the machine code programmer. Let's look at the program again, with location addresses included, as if it were to be loaded at $0000, and the accumulated result to be stored at address $0009. (This is purely for example's sake: any attempt to use these particular locations would almost certainly result in an unrecoverable crash). The two versions of the program are:

Location Address	Machine Code	Assembly Language
Z80		
0000	A7	AND A
0001	3E 42	LD A,$42
0003	CE 07	ADC A,$07
0005	32 09 00	LD BYTE1,A
0008	C9	RET
6502		
0000	18	CLC
0001	A9 42	LDA #$42
0003	69 07	ADC #$07
0005	8D 09 00	STA BYTE1
0008	60	RTS

Note that the fourth instruction (which stores the accumulator contents at $0009) in both programs does not specify a destination address in the Assembly language column. Instead, it uses a symbolic address, BYTE1. In the machine code version of the instruction, however, we see the op-code for 'transfer the accumulator contents' followed by 09 00, the two-byte lo-hi form of the address $0009.

This is another aspect of the translation (or assembly) process from Assembly language to machine code. Just as we use reasonably meaningful instruction mnemonics (STA and RET, for example, instead of hex codes such as 8D and C9) because they make the programs easier for us to read and write, so we will often use symbols such as BYTE1 instead of unfriendly hex addresses or numbers like $0009. This is no different from initialising variables with constants in a BASIC program, and the reasoning is exactly the same in both cases — the program is made more readable, the possibility of errors occurring when writing such numbers is reduced, and the program is made more easily manipulable. For example, changing the statement in which the constant value is assigned to the variable in the first place will cause the new value to be used throughout the program automatically, needing no further editing on the programmer's part.

This is easy to understand when talking about BASIC programming, but where in our Assembly language program is the equivalent of the BASIC statement LET BYTE1=$0009? At present there isn't any such instruction. When we actually come to assemble the Assembly language into machine code we must remember to do this ourselves. If, however, we were using an assembler program to do the assembly for us, then we could make such an assignment statement at the beginning of the program (we give the Z80 version of the program here, although these assembler directives may be used with both processors):

Z80		
0000		BYTE1 EQU $0009
0000	A7	AND A
0001	3E 42	LD A,$42
0003	CE 07	ADC A,$07
0005	32 09 00	LD BYTE1,A
0008	C9	RET

BYTE1 is placed in a column of its own, known as the label field, which we will say more about later. In the op-code field, a new mnemonic (EQU, standing for 'equate' or 'is to be set equal to...') is used; and, in the operand field, the value that is to be assigned to BYTE1 is given (in this case, $0009).

It is important to note that although EQU appears in the op-code field, and looks like a mnemonic, it isn't an Assembly language mnemonic and doesn't belong in either the Z80 or 6502 instruction sets. Such a mnemonic is called a *pseudo-op* or an *assembler directive*. EQU tells the assembler program that 'whenever it finds the preceding alphanumeric symbol (BYTE1 in this case), it must replace it by the value that follows the directive ($0009 here)'. Remember that when we use an assembler program we write only the Assembly language program, either as a tape/disk file or directly at the keyboard, and then call the assembler program to turn it into a machine code program. The output of an assembler is usually a full Assembly listing like those we've been producing, plus the machine code program consisting simply of a string of hex

bytes. The machine code can be saved as a file for later use, or loaded immediately into memory for execution.

In performing the assembly for us, the assembler can be made to perform other tasks that we've been doing by hand — attaching the location addresses to each line of the program, for example. Another pseudo-op, ORG, does this for us. It is added to the program like this:

Z80			
0000		ORG	$A000
A000		BYTE1 EQU	$0009
A000	A7	AND	A
A001	3E 42	LD	A,$42
A003	CE 07	ADC	A,$07
A005	32 09 00	LD	BYTE1,A
A008	C9	RET	

Notice that the location address attached to the first line of the program is $0000, but the address of the following line is $A000, which reflects the effect of the ORG statement. Furthermore, notice that no machine code bytes appear on the lines containing pseudo-ops, precisely because they are not parts of the program and are not to be translated into machine code. Because they are features of the assembler program rather than elements of the CPU instruction set, pseudo-ops do differ from one assembler program to another. EQU, for example, is sometimes replaced by '=', and ORG by '.='. The effect is the same, however, and we shall continue to use ORG and EQU as if they were standard.

It may have occurred to you, while reading about assembler directives, that we've been using a pseudo-op almost from the start of the series: '$', the hex marker. This is no more than a directive to the assembler that what follows is to be treated as a hexadecimal number. Similarly, '#', introduced in the last instalment, is the 'immediate data' marker, signifying that what follows is an absolute quantity rather than a pointer or a symbol. Taking this a little further we could in fact regard Assembly language itself as no more than a series of pseudo-ops. Indeed, there's nothing to stop you inventing your own mnemonics for the machine code instruction set, provided they correspond one-to-one with that set. One very popular assembler program for the Vic-20 does just that: it uses a non-standard version of 6502 Assembly language, largely for the sake of formatting the Vic's 22-column screen.

In this course, we shall continue to use what we've been using so far — the Assembly language mnemonics published by the chip manufacturers — but it does no harm to be reminded from time to time that everything that we call machine code is symbolic. The CPU is indifferent to everything except voltage patterns on its input/output pins, so how we describe those patterns is entirely a matter of convention.

Having finished with pseudo-ops for the moment, let's return to inspecting our program for other points of interest. In particular, let's compare the translations here to the LDA and LD A instructions with their translations in our earlier programs. Previously we wrote:

```
AD ?? ??    LDA $????          (6502)
3A ?? ??    LD  A,($????)      (Z80)
```

meaning 'load the accumulator from the byte whose address is ????'. The load-the-accumulator op-code in translation is $AD (6502) and $3A (Z80). Compare this with the second line of the current program, which incorporates the instruction 'load the accumulator with the immediate value $42'. Here the op-codes are $A9 and $3E, for the 6502 and Z80 respectively. But why are they different? Possibly you've figured it out for yourself. Although we're doing the same class of operation (transferring data into the accumulator) in both programs, the source of that data differs. Therefore, to the CPU, they're different operations, and have different op-codes.

In the first version, data is to be loaded from a byte whose address is given. Nothing is stated or implied about the contents of that byte; the CPU is instructed simply to copy those contents into the accumulator. The three machine code bytes, AD ?? ?? and 3A ?? ??, are decoded by the CPU to mean 'interpret the two bytes following this op-code as the absolute address of the data source'.

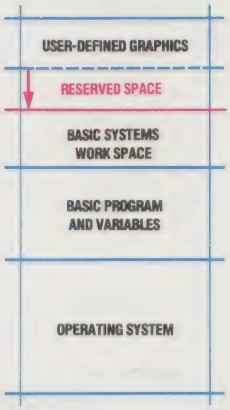
In the second version, the data to be loaded into the accumulator is actually in the byte following the op-code, so the two machine code bytes, A9 42 and 3E 42, are decoded by the CPU to mean 'interpret the byte following this op-code as the data source'. Something in the op-code (A9 or 3E) tells the CPU to load the accumulator from the next byte. Since its program counter always contains the address of the next instruction to be executed, the CPU can calculate the address of the source byte, and then do a simple 'load the accumulator from an addressed byte' operation.

This reinforces the point that the operations of the CPU are mostly very simple, uncomplicated procedures. One whole class of its operations (about a fifth of its entire repertoire) consists of operations that involve copying data from an addressed byte into one or other of its internal registers. These 'primitive' operations all involve one task — to transfer data from memory to a CPU register — and all that distinguishes one instruction from another is the format in which the address of the source byte is presented.

Digging this deep into CPU micro-operations is potentially confusing at first, but well worthwhile for the unifying insights it brings later. Such insights are unnecessary if all you want to do is write Assembly language programs for the sake of speed and efficiency. To do that you need only pick up the idea, learn the instruction set, get a few programming tips, and start right in. If you want to understand what you're doing, however, you'll want to do more than just add another programming language to your range, and you'll find that understanding how one processor works makes learning other Assembly languages enormously easier and more interesting.

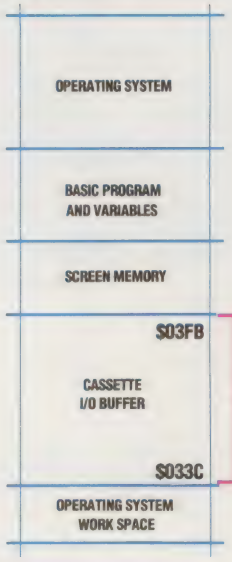
Spectrum

In direct mode enter:
LET RTOP=PEEK (23730) + 256*PEEK (23731)
LET RTOP=RTOP-N
PRINT "RTOP= ";RTOP
where N is the number of bytes you wish to reserve for your program. Your reserved space starts at 1+RTOP



Commodore 64

Use the cassette buffer at $033C to $03FB



Warning
You must adjust these memory pointers immediately after turning on your machine when there is no BASIC program in memory

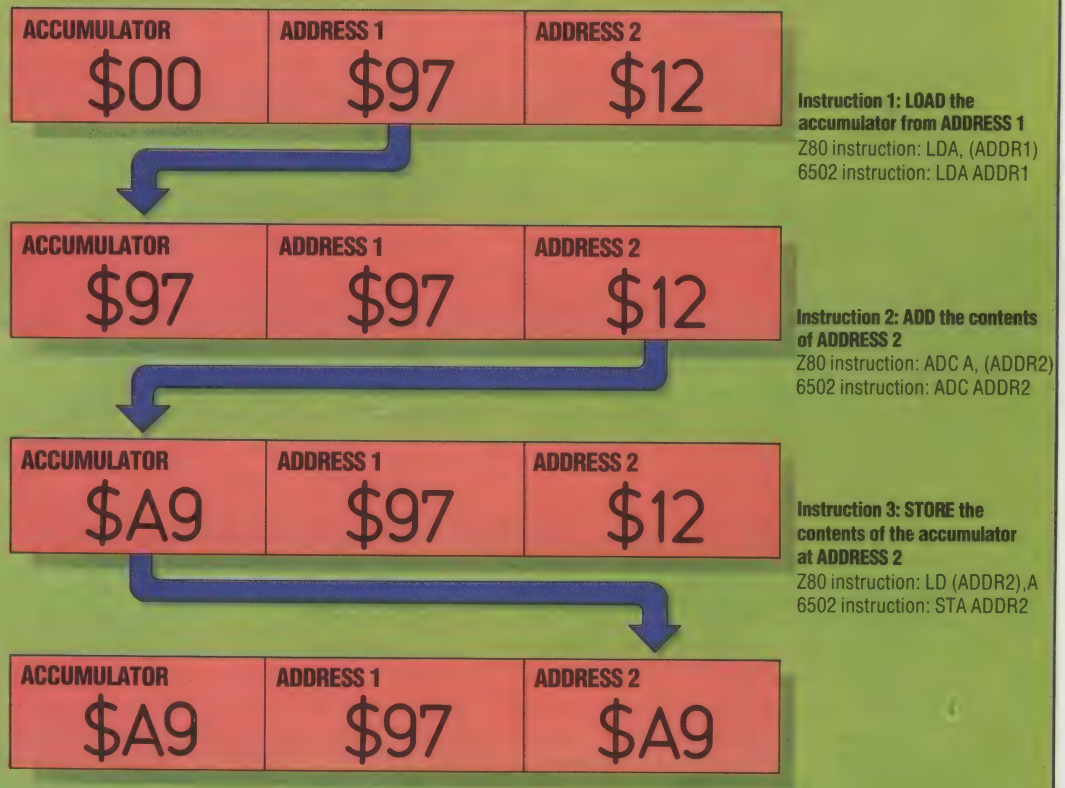
Assembly Exercise

1) In the box on the right, we have given the Assembly language version of a simple program. Assemble the program into machine code and determine the location addresses
2) What instruction is missing from this program?
3) What is the effect of the program on the registers and RAM concerned?
4) What does the term 'immediate data' mean? What other kinds of data could there be?
5) If BYTE1 in the program is treated as an address, on which page of RAM does it appear?

N.B. The values given in this program are for example only: if you wish to execute it, then you must choose locations and values suitable to your machine

Location Address	Machine Code	Assembly Language		
6502				
		START	EQU	$A000
		BYTE1	EQU	$45
		BYTE2	EQU	$38
			ORG	START
			LDA	#BYTE1
			CLC	
			ADC	#BYTE1
			STA	BYTE1
			ADC	#BYTE2
			STA	BYTE2
Z80				
		START	EQU	$A000
		BYTE1	EQU	$45
		BYTE2	EQU	$38
			ORG	START
			LD	A,BYTE1
			AND	A
			ADC	A,BYTE1
			LD	(BYTE1),A
			ADC	A,BYTE2
			LD	(BYTE2),A

The Effect Of Machine Code Instructions



The effect of data transfer instructions, such as LDA ADDR1 or LD (ADDR2),A, is always to copy the contents of the source location into the destination location. This location's contents are therefore overwritten; the source location is unaffected by the data transfer

LEADING EDGE

Probably the most interesting of all Japanese computer makers is also one of the least well-known — SORD. Most of the familiar names, from Hitachi to Sony, are all huge corporations employing thousands of people and with enormous resources, but SORD is a small company with only a few hundred employees.

SORD derives its curious name from a combination of SOftware and haRDware. This is very fitting, since the company has always paid as much attention to the development of software as it has to machinery.

The company started somewhat falteringly. Its president, Takayoshii Shiina, left university to join Rikei Industries, a moderately successful company in the second rank of the Tokyo Stock Exchange in 1967. There he set about reorganising the company's marketing policy.

That Shiina was employed at Rikei at all is a small miracle in a land where people are taken on for life. He apparently announced in his interview that, since he intended to start his own business, he would only be staying for a few years.

By 1970, Shiina and a friend were ready to set up their own company, and so SORD was formed with a capital of 650,000 yen (under £2,000 at today's exchange rate). Shiina, however, continued to work for Rikei until December of that year. SORD's first products consisted of a low-cost logic tester and some contract programming work.

By 1971 SORD was beginning to pull in a reasonable amount of business, still mainly in writing software to order. By 1973 SORD had started manufacturing and by the end of 1974 had an imported floppy disk drive working with a SORD developed interface. This was soon followed by the SMP-80/20, one of Japan's first Intel 8080-based computers.

The SMP-80/20 was a very successful product and orders poured in. But Shiina had big ideas for expansion and in 1977 brought in Toppan, one of Japan's biggest printing companies, for a 20 per cent stake. The cash injection this provided helped SORD to develop considerable software support to go with its growing list of computer products, and by 1981 it had developed PIPS.

PIPS was a software package rather ahead of its time. It was one of the first examples of integrated software in the world and, more than any other factor, helped cement SORD's position in the marketplace. PIPS combines the functions of a spreadsheet, word processor and database in a form that even people who have never used a computer before can learn to use in a few hours.

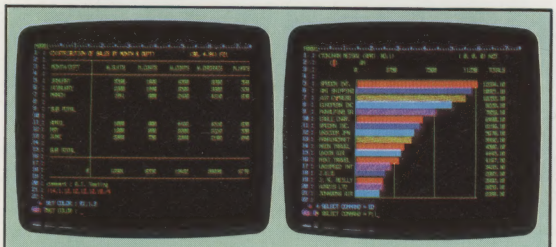
COURTESY OF SORD

Takayoshii Shiina

The phenomenal success of PIPS in Japan can only be understood against the background of the domestic computer market of the time. In Japan it was standard practice for computers to be sold without supporting software.

Applications software of the type familiar in Europe and America for accounting, invoicing and so on was virtually unheard of in Japan. The reasons were twofold. Firstly, the Japanese reluctance to import anything they can do for themselves meant that very little American software was coming in to the country. Secondly, the Japanese are no linguists, ranking in aptitude only slightly above the British. If a Digital Research manual on CP/M is arcane to a British reader, it is totally incomprehensible to a Japanese reader. The dearth of applications software in the Japanese market left a gap wide open for SORD to exploit.

SORD started to develop the M200 series of Z80-based computers and, later the M23 series. Larger, multi-user requirements were satisfied with the M343 series. The M5 was introduced to cater for the home games market, and now we have the M68 series, incorporating both Z80 and Motorola 68000 processors, the M243 small



PIPS
PIPS (Pan Information Processing System) is SORD's own unique business package. It is a general-purpose program that can be set up to handle most business tasks from accounting to presentation graphics. It demonstrates a curious Japanese approach to marketing as it is only available free with certain SORD machines. This may provide a good reason to buy a SORD, but it also limits the popularity of an exciting and useful product

business computers, and the M285 32-bit computer running VAX-11 software for CAD (computer aided design) applications.

No other company in the world has such a wide range of computer products on the market, and every one comes with a supporting set of software. Where SORD seems to fail is that it sees its products in 'turn-key' terms. That is to say, it sees its machines as fully working systems, complete with hardware and software. SORD users do not generally have the option of choosing other software products for their machines.

The company went some way towards rectifying this situation by supplying the SB-80 operating system as an option to go with its M23 series of Z80-based computers. SB-80 is equivalent to Digital Research's CP/M 2.2 operating system and it allows CP/M software to be used, for the first time, on SORD computers.

SORD's other step in this direction was to make the UCSD (University of California at San Diego) p-System available.

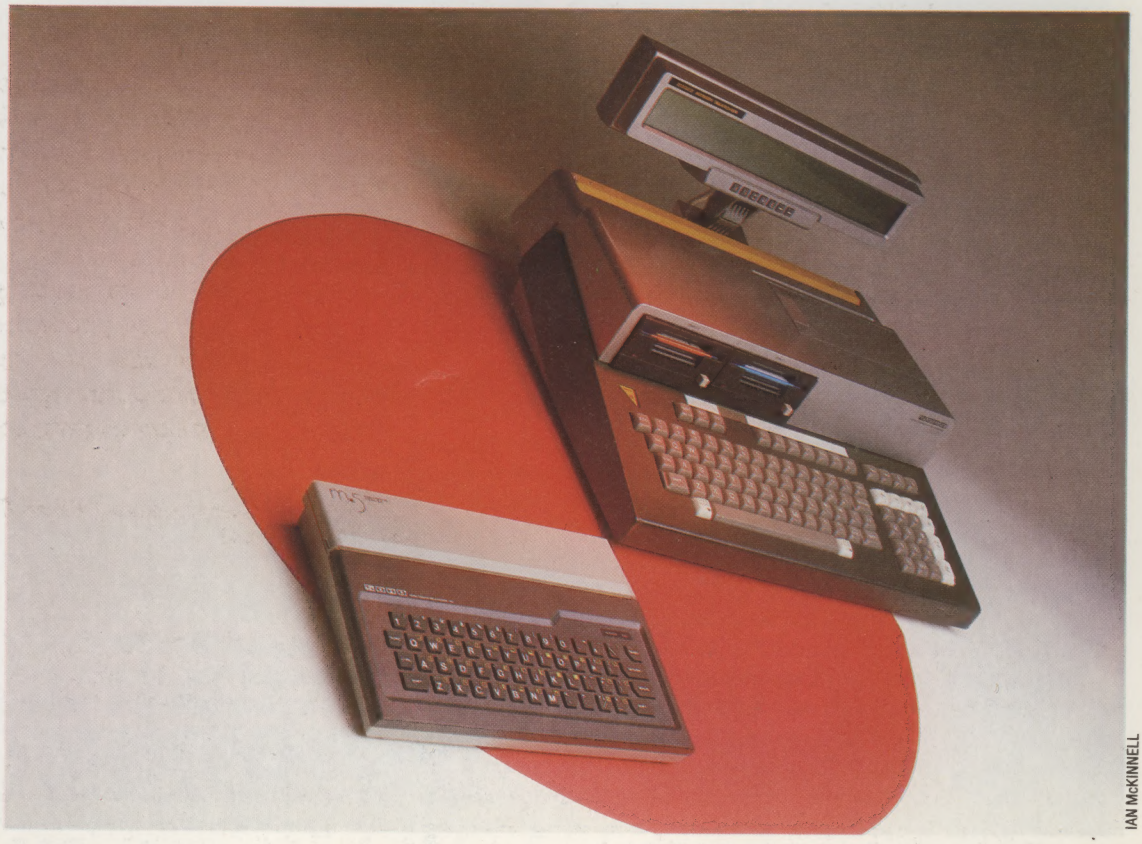
For users content to stick with SORD's own software, the company offers several powerful BASICS, its own operating system, PIPS, and a Wang lookalike word processor.

Apart from the CP/M and p-system offerings, SORD has tended to tread the path beaten by minicomputer manufacturers such as DEC, by offering systems that depend largely on in-house software for their operation. Meanwhile, the rest of the computer world has found that standardisation and inter-company compatibility is the way to generate sales. SORD has proved itself, so far, slow to adapt to this trend.

Meanwhile, SORD continues to come up with innovative hardware designs but, critics say, fails to support customers with adequate documentation for its products. Over the last two years SORD has made gargantuan efforts to provide good documentation for both PIPS and its word processor. In-depth documentation of its hardware still seems to be lacking, however, and there is still a reluctance to encourage third party software houses to develop products to run on its computers.

The next few years will probably be critical for SORD, faced as it is with the industrial might of the larger Japanese computer firms and of IBM. SORD's Shiina firmly believes in small-scale enterprise and individualism, and he has succeeded in making SORD an international company. Will it be able to stay ahead in the competitive computer world of today?

SORD Options
SORD's best-known machines are its home computer, the M5, and its portable M23P business machine. The M23P set new standards for portables by using Sony floppy disks and an 80-column LCD



IAN McKINNELL

GET MACHINE CODE TAPED WITH CHAMP

* CHAMP: A uniquely comprehensive aid to machine code programming
* Specially commissioned to accompany the *Home Computer Advanced Course* series on machine code
* Suitable for the BBC Model B, Commodore 64 and 48K Sinclair Spectrum
* Pre-recorded on tape cassette FREE with Issue 11 of *The Home Computer Advanced Course*

To unlock the full potential of your computer you have to talk to it in its own language — machine code. The *Home Computer Advanced Course* is teaching you to do just that in its unrivalled series on machine code programming. And now you can bring the computer itself to your aid. With the pre-recorded software that is being given away with Issue 11, you can tackle ambitious programs — thanks to these features, rarely found together in one package:

The Editor enables you to enter your programs, written in Assembly language mnemonics, from the keyboard, or to load them from tape. You can then modify the program freely, with on-entry syntax checking.

The Assembler translates your Assembly language program into machine code — saving you hours of tricky, error-prone work

The Monitor displays contents of memory, together with hex and ASCII equivalents, and enables you to move blocks of memory, search for specified strings, and modify the memory contents directly

The Disassembler translates the machine code version of the program back into Assembly language mnemonics for your scrutiny

The Debug Facility enables you to run the program step by step while displaying the contents of the microprocessor's registers and modifying them as necessary

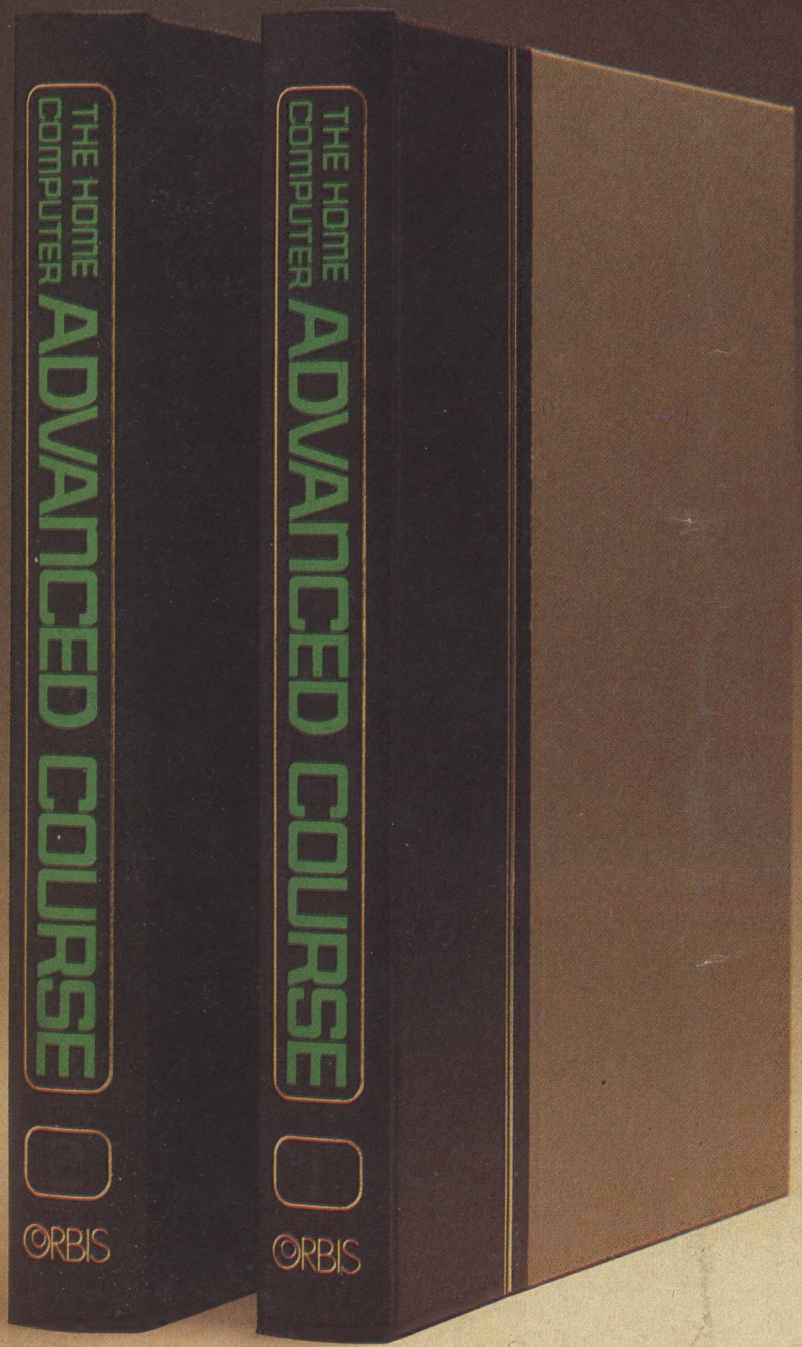
Normally you'd have to buy several different software packages to obtain all these facilities — and you'd have to spend pounds. They're *free* with Issue 11 of *The Home Computer Advanced Course*. This suite of advanced programs was specially commissioned from a leading software company, Personal Software Services, to be a valuable programming aid for readers. With its aid you'll learn how to make your micro do things that will amaze you: seemingly instantaneous program execution and dazzling, lightning-fast graphics animation.

(CHAMP — Comprehensive Home Computer User's Assembler/ Monitor Package)

GET MACHINE CODE TAPED WITH ISSUE 11 OF THE HOME COMPUTER ADVANCED COURSE

THE HOME COMPUTER ADVANCED COURSE

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.



All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.

Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.

The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

PLACE A STANDING ORDER TODAY.